НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА СИСТЕМНОГО ПРОГРАМУВАННЯ І
СПЕЦІАЛІЗОВАНИХ КОМП'ЮТЕРНИХ СИСТЕМ

«На правах рукопису»
УДК _____

«До захисту допущено»
Завідувач кафедри СПСКС

_____ В.П.Тарасенко_
(підпис) (ініціали, прізвище)
"___"_____2018р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 123 Комп'ютерна інженерія (Комп'ютерні системи та компоненти)

на тему: _____
_____
_____
_____

Виконав (-ла): студент (-ка) ІІ курсу, групи _КВ-61_м_____
(шифр групи)

_____ _____
(прізвище, ім'я, по батькові) (підпис)

Науковий керівник _____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент_____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

КИЇВ – 2018

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра системного програмування і спеціалізованих комп'ютерних систем

Рівень вищої освіти – другий (магістерський)

Спеціальність 123 Комп'ютерна інженерія (Комп'ютерні системи та компоненти)

ЗАТВЕРДЖУЮ
Завідувач кафедри СПСКС
_____ _В.П.Тарасенко__
(підпис) (ініціали, прізвище)
«___»_____2018р.

ЗАВДАННЯ

на магістерську дисертацію студенту

Бондарчуку Максиму Юрійовичу

1. Тема дисертації: метод рою частинок пошуку найкоротшого шляху в телекомунікаційній мережі, науковий керівник дисертації к.т.н., доцент Зорін Юрій Михайлович, затверджені наказом по університету від «22» березня 2018 р. №986-с.

2. Термін подання студентом дисертації 11 травня 2018 р.

3. Об'єкт дослідження: процес комбінаторної оптимізації при розв'язанні задачі пошуку найкоротших шляхів в телекомунікаційній мережі.

4. Предмет дослідження: евристичні методи розв’язання задачі пошуку найкоротших шляхів в телекомунікаційній мережі

5. Перелік завдань, які потрібно розробити:
    - Виконати огляд існуючих алгоритмів пошуку найкоротшого шляху в телекомунікаційній мережі.
    - Виконати огляд існуючих метаевристичних алгоритмів.
    - Розробити модифікований метод рою частинок.
    - Застосувати розроблений метод до задачі пошуку найкоротшого шляху в телекомунікаційній мережі.

6. Перелік ілюстративного матеріалу: 22 рисунка, 3 схеми, 6 таблиць.

7. Перелік публікацій
    - Наукова конференція магістрантів та аспірантів «Прикладна математика та комп’ютинг» ПМК-2017 (Київ, 19-21 квітня 2017 р.)
    - Наукова конференція магістрантів та аспірантів «Прикладна математика та комп’ютинг» ПМК-2018 (Київ, 13-15 квітня 2018 р.).

8. Дата видачі завдання 5 вересня 2016 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Розробка технічного завдання	10.01.2018	
2	Аналіз існуючих рішень	05.03.2018	
3	Розробка методу рою частинок пошуку найкоротшого шляху в телекомунікаційній мережі	25.03.2018	
4	Вибір середовища розробки	26.03.2018	

5	Розробка програмного продукту	30.03.2018	
6	Тестування запропонованого методу	01.04.2018	
7	Підготовка пояснювальної записки	14.04.2018	
8	Оформлення матеріалів проекту	15.04.2018	
9	Попередній розгляд магістерської дисертації на кафедрі	26.04.2018	

Студент _____ _____

(підпис) (ініціали, прізвище)

Науковий керівник дисертації _____ _____

(підпис) (ініціали, прізвище)

РЕФЕРАТ

Актуальність теми. В наш час важко уявити життя без телекомунікаційних мереж. Мережа інтернет, яка є телекомунікаційною мережею, розширюється з кожним днем долучаючи тисячі нових користувачів. Україна є однією з країн, в якій інтернет розвивається швидкими темпами. На 2017 рік у світі нараховувалось близько 4 мільярдів користувачів всесвітньої мережі або 50% населення Землі. Значно зростає кількість користувачів соцмереж і частка мобільних пристроїв у мережі. Незважаючи на те, що майже всі куточки планети покриті мережею, у багатьох місцях все ще наявна незначна пропускна спроможність мережі. В цьому випадку ефективне розв'язання задачі пошуку найкоротших шляхів між заданими вузлами є надзвичайно важливою задачею.

Об'єктом дослідження є процес комбінаторної оптимізації при розв'язанні задачі пошуку найкоротших шляхів в телекомунікаційній мережі.

Предметом дослідження є евристичні методи розв'язання задачі пошуку найкоротших шляхів в телекомунікаційній мережі.

Метою роботи є розробка нового методу розв'язання задачі пошуку найкоротших шляхів в телекомунікаційній мережі, що характеризується вищою швидкодією та якістю розв'язків ніж відомі методи.

Методи дослідження. В роботі використовуються методи евристичних алгоритмів, методи дискретної математики, методи комбінаторної оптимізації.

Наукова новизна полягає в наступному:

1. В результаті аналізу метаевристик в якості базового алгоритму було вибрано алгоритм рою частинок.
2. Запропоновано модифікації алгоритму рою частинок.
3. Розроблено метод застосування алгоритму рою частинок до задачі пошуку найкоротших шляхів в телекомунікаційній мережі.

Практична цінність отриманих в роботі результатів полягає в тому, що запропонований метод дає змогу знаходити найкоротші шляхи в телекомунікаційній мережі з більш високим показником успішності (доля знаходження правильного шляху для 100 тестів) ніж у базового алгоритму.

Апробація роботи. Основні положення й результати роботи були представлені та обговорювались на науковій конференції магістрантів та аспірантів «Прикладна математика та комп'ютинг» ПМК-2017 (Київ, 19-21 квітня 2017 р.), а також на науковій конференції магістрантів та аспірантів «Прикладна математика та комп'ютинг» ПМК-2018 (Київ, 13-15 квітня 2018 р.).

Структура та обсяг роботи. Магістерська дисертація складається з вступу, чотирьох розділів та висновків.

У вступі подано загальну характеристику роботи, зроблено оцінку сучасного стану проблеми, обґрунтовано актуальність напрямку досліджень, сформульовано мету і задачі досліджень, показано наукову новизну отриманих результатів і практичну цінність роботи, наведено відомості про апробацію результатів і їхнє впровадження.

У першому розділі розглянуто опис предметної області досліджень та проаналізовано методи розв'язання проблеми.

У *другому розділі* описано базовий алгоритм рою частинок, модифікації алгоритму рою частинок, а також методику його застосування до задачі пошуку найкоротших шляхів в телекомунікаційній мережі.

У *третьому розділі* описано програмний додаток, в якому реалізовано розроблений метод.

У *четвертому розділі* описано результати тестування розробленого алгоритму.

У висновках представлені результати проведеної роботи.

Робота представлена на 103 аркушах, містить посилання на список використаних літературних джерел.

Ключові слова: метаевристичні алгоритми, найкоротший шлях, алгоритм рою частинок, телекомунікаційна мережа.

РЕФЕРАТ

Актуальность темы. В наше время трудно представить жизнь без телекоммуникационных сетей. Сеть интернет, которая является телекоммуникационной сетью, расширяется с каждым днем привлекая тысячи новых пользователей. Украина является одной из стран, в которой интернет развивается быстрыми темпами. На 2017 год в мире насчитывалось около 4 миллиардов пользователей всемирной сети или 50% населения Земли. Значительно возрастает количество пользователей соцсетей и доля мобильных устройств в сети. Несмотря на то, что почти все уголки планеты покрыты сетью, во многих местах все еще имеется незначительная пропускная способность сети. В этом случае эффективное решение задачи поиска кратчайших путей между заданными узлами является чрезвычайно важной задачей.

Объектом исследования является процесс комбинаторной оптимизации при решении задачи поиска кратчайших путей в телекоммуникационной сети.

Предметом исследования является эвристические методы решения задачи поиска кратчайших путей в телекоммуникационной сети.

Целью работы является разработка нового метода решения задачи поиска кратчайших путей в телекоммуникационной сети, характеризующееся высоким быстродействием и качеством решений чем известные методы.

Методы исследования. В работе используются методы эвристических алгоритмов, методы дискретной математики, методы комбинаторной оптимизации.

Научная новизна заключается в следующем:

1. В результате анализа метаалгоритмов в качестве базового алгоритма был выбран алгоритм роя частиц.
2. Предложены модификации алгоритма роя частиц.

3. Разработан метод применения алгоритма роя частиц к задаче поиска кратчайших путей в телекоммуникационной сети.

Практическая ценность полученных в работе результатов заключается в том, что предложенные модификации алгоритма позволяют находить кратчайшие пути в телекоммуникационной сети с более высоким показателем успешности (доля нахождения правильного пути на 100 тестов), чем у базового алгоритма.

Апробация работы. Основные положения и результаты работы были представлены и обсуждались на научной конференции магистрантов и аспирантов «Прикладная математика и компьютинг» ПМК-2017 (Киев, 19-21 апреля 2017), а также на научной конференции магистрантов и аспирантов «Прикладная математика и компьютинг» ПМК-2018 (Киев, 13-15 апреля 2018).

Структура и объем работы. Магистерская диссертация состоит из введения, четырех разделов и выводов.

Во *вступлении* представлена общая характеристика работы, произведена оценка современного состояния проблемы, обоснована актуальность направления исследований, сформулированы цели и задачи исследований, показано научную новизну полученных результатов и практическую ценность работы, приведены сведения об апробации результатов и их внедрение.

В *первом разделе* рассмотрены описание предметной области исследований и проанализированы методы решения проблемы.

Во *втором разделе* описано базовый алгоритм роя частиц, модификации алгоритма роя частиц, а также метод его применения к задаче поиска кратчайших путей в телекоммуникационной сети.

В *третьем разделе* описано программное приложение, в котором реализован метод.

В *четвертом разделе* описаны результаты тестирования разработанного метод.

В *выводах* представлены результаты проведенной работы.

Работа представлена на 103 листах, содержит ссылки на список использованных литературных источников.

Ключевые слова: метаэвристические алгортмы, кратчайший путь, алгоритм роя частиц, телекоммуникационная сеть.

ABSTRACT

Actuality of theme. In our time, it's hard to imagine a life without telecommunication networks. The Internet, which is a telecommunication network, is expanding every day by engaging thousands of new users. Ukraine is one of the countries in which the Internet is developing rapidly. By 2017, the world has about 4 billion users worldwide or 50% of the Earth's population. The number of social network users and the share of mobile devices in the network is growing significantly. Even though almost all the corners of the planet are covered by the network, in many places there is still a small bandwidth of the network. In this case, an effective solution to the problem of finding the shortest paths between the given nodes is an extremely important task.

The object of the study is the process of combinatorial optimization in solving the problem of finding the shortest paths in the telecommunication network.

The subject of the research is the heuristic methods of solution of shortest path problem in the telecommunication network.

The aim of the work is to develop a new method for solving the problem of finding the shortest paths in a telecommunication network characterized by higher speed and quality of solutions than known methods.

Research methods. Methods of heuristic algorithms, methods of discrete mathematics, methods of combinatorial optimization are used in this work.

The scientific novelty is as follows:

1. As a result of the analysis of meta-heuristics, the algorithm of particle swarm was chosen as the basic algorithm.
2. Modifications of the particle swarm optimization algorithm are proposed.
3. The method of using the particle swarm algorithm for the problem of finding shortest paths in a telecommunication network is developed.

The practical value of the results obtained in the work is that the proposed modifications of the algorithm provide an opportunity to find shortest paths in a

telecommunication network with a higher success rate (the part of finding the correct path for 100 tests) than in the basic algorithm.

Approbation. The main provisions and results of the work were presented and discussed at the scientific conference of masters and postgraduates "Applied Mathematics and Computing", ПМК-2017 (Kyiv, April 19-21, 2017), as well as at the scientific conference of undergraduate and postgraduate students "Applied Mathematics and Computing » ПМК-2018 (Kyiv, April 13-15, 2018).

Structure and scope of work. The master's dissertation consists of an introduction, four sections and conclusions.

The introduction gives a general description of the work, assesses the current state of the problem, substantiates the relevance of the research direction, formulates the purpose and objectives of the research, shows the scientific novelty of the results obtained and the practical value of the work, provides information on the approbation of the results and their implementation.

The first section describes the description of the subject area of research and analyzes the methods of solving the problem.

The second section describes the basic algorithm of particle swarm, modification of the particle swarm algorithm, as well as the method of its application for the task of finding the shortest paths in the telecommunication network.

The third section describes a software application that implements the developed algorithm.

The fourth section describes the results of testing the developed algorithm.

The conclusions are the results of the work.

The work is presented on 103 sheets, contains a link to the list of used literary sources.

Keywords: metaheuristic algorithms, shortest path, particle swarm optimization algorithm,, telecommunication network.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

ABC (Artificial Bee Colony) – алгоритм бджолиної колонії.

AJAX (asynchronous JavaScript and XML) – підхід до побудови користувацьких інтерфейсів web-додатків.

AOT (ahead-of-time compilation) – компіляція перед виконанням.

API (application programming interface) – прикладний програмний інтерфейс.

CIL (Common Intermediate Language) – загальна проміжна мова.

CLI (Common Language Infrastructure) – специфікація міжмовної інфраструктури.

CLR (Common Language Runtime) – вихідний файл виконання мов фреймворку .NET.

COM (Component Object Model) – платформа компонентно-орієнтованого програмування.

CRUD (Create Read Update Delete) – набір базових операцій з базою даних: create, read, update, delete.

CTS (Common Type Specifications) – специфікації міжмовних типів фреймворку .NET.

DLL (Dynamic-link library) – динамічно приєднувана бібліотека

ECMA (European Computer Manufacturers Association) – некомерційна асоціація європейських виробників комп’ютерів.

EXE (executable) – розширення виконуваного файлу.

FCL (Framework Class Library), BCL (Base Class Library) – стандартна бібліотека класів платформи.

GA (genetic algorithm) – генетичний алгоритм.

IL – intermediate language – проміжна мова фреймворку .NET.

ISO (International Organization for Standardization) – міжнародна організація зі стандартизації.

JIT (just in time [compiler]) – компілятор миттєвого виконання

FFA (Firefly Optimization Algorithm) – алгоритм оптимізації світлячків.

LINQ (Language Integrated Query) – інтегровані запити мови C#.

PSO (Particle Swarm Optimization) – алгоритм рою частинок.

SPP (shortest path problem) – задача пошуку найкоротшого шляху.

SSL (Secure Sockets Layer) – рівень захищених сокетів.

XML (Extensible Markup Language) – мова розмітки ієрархічно структурованих даних.

UWP (Universal Windows Platform) – універсальна платформа Windows.

ВСТУП

В наш час важко уявити нормальне людське життя без телекомунікаційних мереж. Мережа інтернет, яка є телекомунікаційною мережею, розширюється з кожним днем долучаючи тисячі нових користувачів. Україна є однією з країн, в якій інтернет розвивається швидкими темпами. На 2017 рік у світі нараховувалось близько 4 мільярдів користувачів всесвітньої мережі або 50% населення Землі. Значно зростає кількість користувачів соцмереж і частка мобільних пристроїв у мережі. Незважаючи на те, що майже всі куточки планети покриті мережею, у багатьох місцях все ще наявна незначна пропускна спроможність мережі. В цьому випадку ефективне розв’язання задачі пошуку найкоротших шляхів між заданими вузлами є надзвичайно важливою задачею.

Задача пошуку найкоротших шляхів в телекомунікаційних мережах полягає в знаходженні такого шляху між вузлами, передача повідомлень через який буде оптимальною за однією з характеристик. Для методу рою частинок цією характеристикою є мінімальна відстань з врахуванням пропускної спроможності каналів передачі даних. В загальному, дана задача зводиться до задачі знаходження найкоротшого шляху в графі. Існує багато алгоритмів пошуку найкоротших шляхів. Деякі з алгоритмів відносяться до теорії графів (наприклад, алгоритм Дейкстри). Інші ж алгоритми є метаевристичними, тобто містять у собі набір алгоритмічних понять, які можуть бути використані для визначення евристичних методів, що застосовуються для широкого кола різних завдань. Використання метаевристики значно підвищує можливість знаходження якісного розв’язку для складних, актуальних комбінаторних задач оптимізації за розумний час.

В роботі розроблено модифікований метаевристичний метод рою частинок для розв’язання задачі пошуку найкоротших шляхів у

телекомунікаційних мережах з досить високими показниками в сенсі якості розв’язку, швидкості збіжності й стабільності результатів.

МЕТА ДОСЛІДЖЕННЯ ТА ПОСТАНОВКА ЗАДАЧІ

Метою дослідження в роботі є розробка модифікованого алгоритму рою частинок для розв’язання задачі пошуку найкоротших шляхів у телекомунікаційних мережах.

Наукова задача, що розв’язується в даній роботі включає наступні завдання:

1. Аналіз алгоритмів пошуку найкоротших шляхів.
2. Аналіз метаевристичних алгоритмів.
3. Аналіз базового методу рою частинок.
4. Розробка модифікованого методу рою частинок.
5. Розробка методики застосування модифікованого методу рою частинок для задачі пошуку найкоротших шляхів у телекомунікаційних мережах.

Для практичного підтвердження результатів розробленого методу необхідно створити програму, що дозволяє виконувати оригінальний та модифікований методи рою частинок.

1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ДЛЯ РОЗВ'ЯЗАННЯ ЗАДАЧІ ПОШУКУ НАЙКОРОТШИХ ШЛЯХІВ ТЕЛЕКОМУНІКАЦІЙНІЙ У МЕРЕЖІ

На даний момент існує доволі велика кількість алгоритмів пошуку найкоротших шляхів в телекомунікаційних мережах. Оскільки для даної задачі використано метаевристичний метод, то варто також виконати огляд метаевристичних методів.

1.1. Алгоритми пошуку найкоротших шляхів

1.1.1. Алгоритм Флойда-Воршелла

Алгоритм Флойда-Воршелла є відомим завдяки його застосуванню для розв'язання задачі про найкоротший шлях у зваженому графі з вагами ребер, що додатними або від'ємними (але без від'ємних циклів). Алгоритм має знайти довжини (сумарні ваги) найкоротших шляхів між всіма парами вершин. Проте на цьому все, алгоритм не може вказати конкретні шляхи. Дещо видозмінені версії алгоритму також використовуються для знаходження транзитивних замиканнь у відношенні R, або (враховуючи Метод Шульце), для знаходження найдовших шляхів (англ. widest path problem) між всіма парами вершин у зваженому графі.

Алгоритм Воршелла працює за допомогою порівнянь всіх можливих шляхів у графі між кожною з пар вершин. Вартістю виконання є $O(|V|^3)$ порівнянь. Це доволі поганий результат, враховуючи, що в графі може бути до $O(|V|^2)$ ребер (всі вершини з'єднані між собою), і тоді кожну таку комбінацію буде перевірено. Алгоритм виконує це шляхом поступового покращення вартості найкоротшого шляху між двома вершинами, поки вартість не стає оптимальною.

Для прикладу, розглянемо граф G з ребрами V, пронумерованими від 1 до V. Крім того розгляньмо функцію $shortestPath(i, j, k)$, яка повертає

найкоротший шлях від i до j, за допомогою використання вершин з множини $\{1,2,\dots,k\}$ як внутрішніх для шляху. Тепер, маючи таку функцію, нам потрібно знайти найкоротший шлях від кожної вершини i до кожної вершини j, використовуючи тільки вершини від 1 до $k+1$.

Для кожної з цих пар вершин, найкоротшим шляхом може бути один з наступних:

1. Шлях, у який містить лише вершини з множини $\{1,\dots,k\}$.
2. Шлях, який проходить від i до $k+1$, а потім від $k+1$ до j.

Найкоротший шлях від i до j, що використовує тільки вершини $\{1,\dots,k\}$ можна визначити за функцією $shortestPath(i,j,k)$, і якщо є коротший шлях від i до $k+1$, то довжина цього шляху буде сумою (злиттям) найкоротшого шляху від i до $k+1$ (використовуючи вершини $\{1,\dots,k\}$) і найкоротшого шляху від $k+1$ до j (так само використовуючи вершини з $\{1,\dots,k\}$).

Від'ємний цикл на графі – це цикл, в якому сума всіх його ребер є меншою за нуль. Між парами вершин i та j, між якими є від'ємні ребра не може бути найкоротшого шляху, бо шлях між ними тоді може бути нескінченно малий. Для отримання результату, алгоритм Флойда передбачає відсутність від'ємних циклів. Незважаючи на це, якщо в графі є від'ємні цикли, алгоритм Флойда-Воршалла може бути застосований для їх виявлення. Інтуїтивно це можна виявити наступним чином:

- алгоритм Флойда-Воршалла багаторазово змінює довжини шляху між усіма парами вершин (i,j), включаючи ті, де $i=j$;
- на початку довжина шляху $(i,i)=0$;
- шлях $\{(i,k),(k,j)\}$ може тільки тоді результат, якщо він має довжину меншу за нуль, тобто позначає від'ємний цикл;

- таким чином, після виконання алгоритму, шлях (i, i) буде від'ємним, за умови що існує шлях від'ємної довжини від i назад до i.

Отже, для виявлення від'ємних циклів за допомогою алгоритму Флойда-Воршелла, можна перевірити діагональ матриці шляхів, і наявність від'ємного числа означає, що даний граф містить хоча б один від'ємний цикл. Щоб уникнути проблем, потрібно у внутрішньому циклі перевіряти діагональ матриці шляхів. Очевидно, що в неорієнтованому графі від'ємне ребро створює від'ємний цикл за участю обох інцидентних вершин [1].

1.1.2. Алгоритм Дейкстри

Алгоритм Дейкстри, за допомогою якого можна знайти найкоротший шлях між будь-якими двома вершинами графа, на відміну від алгоритму Флойда-Воршелла, своїм призначенням має пошук найкоротшого шляху від заданої вершини графа до всіх інших вершин. В процесі виконання алгоритму Дейкстри для переходів з вершини i до вершини j застосовується спеціальна процедура, що присвоює відповідну мітку кожній вершині графа.

Нехай за допомогою алгоритму Дейкстри необхідно знайти найкоротшу відстань від початкової вершини 1 до вершини i. Тоді мітка для вершини j в алгоритмі Дейкстри визначатиметься наступним чином:

$$[u_j, i] = [u_i + d_{ij}, i], d_{ij} \geq 0.$$

Мітки вершин алгоритму можуть бути двох типів: тимчасові та постійні. Тимчасова мітка може бути замінена новою (теж тимчасовою), якщо в процесі виконання алгоритму буде знайдено більш короткий шлях з початкової вершини до даної. Якщо ж за час виконання алгоритму виявиться, що більш короткого шляху з початкової до даної вершини не існує, статус тимчасової мітки буде замінено на постійну.

Сам алгоритм складається з наступних кроків:

1. Початковій вершині (вершина 1) присвоюємо постійну мітку $[0, -]$ і встановлюємо $i = 1$.
2. Для всіх вершин j, що з'єднані орієнтованим ребром з вершиною i, і не мають постійних міток, необхідно обчислити тимчасові мітки $[u_i + d_{ij}, i]$. Якщо для вершини j вже існує певна мітка $[u_j, k]$, то необхідно перевірити виконання умови $u_i + d_{ij} < u_j$. Виконання даної умови свідчить про те, що необхідно мітку $[u_j, k]$ замінити міткою $[u_i + d_{ij}, i]$.

Даний процес необхідно продовжувати до того часу, поки всім вершинам графа не буде проставлено постійні мітки [2].

1.2. Метаевристичні алгоритми

1.2.1. Алгоритм бджолиної колонії

Багато зі складних задач оптимізації функції багатьох змінних не можуть бути повністю розв’язані за поліноміально обмежений час. Це явище викликає значний інтерес до алгоритмів пошуку, що за своєю суттю дозволяють знайти оптимальний розв’язок за розумні проміжки часу.

Згідно з біологічними знаннями (а також у реальному житті), бджолина колонія може простягатися на великі відстані (більше 10 квадратних кілометрів) в кількох напрямках й використовуючи в той же самий час значну кількість джерел нектару. Колонія досягає процвітання, за допомогою посилання робочих бджіл на поля з квітами. Є логічним, що, ділянки квітів зі значною кількістю нектару, який можна зібрати докладаючи менших зусиль, повинні бути відвіданими більшою кількістю бджіл, в той час як ділянки з відповідно меншою їх кількістю мають бути відвіданими меншою частиною бджіл.

Збір нектару розпочинається з вулика (колонії), коли бджоли-розвідники відправляються на пошук так званих перспективних ділянок квітів. Ці бджоли рухаються довільним чином, переміщаючись з однієї ділянки квітів на іншу. За час збору нектару колонія продовжує проводити розвідку, завжди тримаючи певний відсоток популяції у ролі бджіл-розвідників.

При поверненні даних бджіл до вулика, ті бджоли-розвідники, які знайшли ділянку, що за своїми якісними характеристиками оцінюється вище певного визначеного порогу (дана величина вимірюється як комбінація певних складових, серед яких основною є вміст цукру), залишають пробу нектару з цього джерела, і виходять на так званий

«танцювальний майданчик» для виконання танцю, що є відомим як «танець з похитуванням».

Цей загадковий в біологічному сенсі танець має важливе значення для обміну інформацією в колонії, і містить у собі три порції інформації щодо квіткової ділянки: напрям, в якому розташована ділянка, відстань даної ділянки від вулика та міра її якості (або фітнес-функція). За допомогою цієї інформації популяція бджіл може відправити своїх робочих бджіл одразу ж до ділянок квітів, без використання для цього жодної іншої інформації. Знання щодо навколишнього середовища кожною бджолою отримується виключно з даного танцю. Танець допомагає колонії виконати оцінку відносного внеску кожної з ділянок за якістю її нектару, та обсягом енергії, що необхідна для збирання нектару з неї. З завершенням виконання цього танцю, розвідник бере з собою бджіл-спостерігачів, що очікують для цієї задачі у вулику, і повертається до квіткової ділянки. Чим більш перспективною є ділянка, тим більше бджіл-спостерігачів буде відправлено до неї. Це дозволяє популяції збирати в більш ефективний та швидкий спосіб.

Колонія бджіл контролює рівень нектару в процесі його збирання з ділянки. Інформація про цей рівень буде використана в наступному танці при поверненні до вулика. У випадку високого показника якості ділянки, вона буде представлена під час наступного танцю й відповідно до неї буде залучено більшу кількість бджіл.

В даному алгоритмі бджолиної колонії власне колонія складається з 3 типів бджіл: робочих, спостерігачів та розвідників. Спостерігачами називають тих бджіл, що очікують біля «майданчика для танців» на прийняття рішення щодо наступного джерела нектару для них. Робочими бджолами називають тих бджіл, що виконують збір нектару з відвіданих ними раніше джерел. Розвідниками називають тих бджіл, що виконують

довільний пошук. Алгоритм ABC поділяє колонію на дві частини, першою з яких є робочі бджоли, а другою є спостерігачі. Кожне джерело нектару обробляється лише однією робочою бджолою. Тобто кількість джерел в околиці вулика є рівною кількості робочих бджіл. Та робоча бджола, джерело якої більше не містить нектару, перетворюється на розвідника.

Алгоритм складається з наступних кроків:

1. Виконати ініціалізацію колонії.
2. Виконувати повторення:
    a. Виконати розміщення робочих бджіл на джерелах нектару.
    b. Виконати розміщення спостерігачів на тих самих джерелах нектару.
    c. Для знаходження нових джерел нектару, відправити розвідників в область пошуку.
3. Повторювати кроки 1-2 допоки не виконані вимоги.

Кожна ітерація пошуку алгоритму ABC містить у собі 3 етапи: робочі бджоли відправляються до джерел нектару та виконують оцінку кількості нектару в даних джерелах; спостерігачі виконують вибір джерел нектару після обміну інформацією з робочими бджолами та визначення кількості нектару в джерелах; визначення, які з робочих бджіл перетворюються на розвідників, після чого виконується їх відправлення до потенційних джерел нектару. На стадії ініціалізації алгоритму бджоли довільним чином обирають джерело харчування та визначають кількість нектару у ньому. Після цього бджоли повертаються до колонії і обмінюються інформацією щодо кількості нектару в квітах з бджолами-спостерігачами, що чекають на танцювальному майданчику. На наступному етапі після обміну інформацією, всі робочі бджола переходять до області джерел нектару, які вони відвідувала на попередніх ітераціях, допоки ці джерела харчування існують в їх пам’яті, після чого бджоли вибирають нові джерела нектару

поміж сусідніх. На останньому етапі спостерігачі визначають найкраще з джерел харчування відповідно до інформації щодо нектару, інформація про який надається робочими бджолами.

Що більшою є кількість нектару в джерелі, тим вищою є ймовірність того, що дане джерело буде обране спостерігачем. Таким чином, танець робочих бджіл, що надають інформацію щодо більшої кількості нектару, приваблює більше спостерігачів до таких ділянок. По прибуттю до обраної ділянки, спостерігач в околиці пам'яті в залежності від візуальної інформації обирає нове джерело нектару. Візуальна інформація отримується відповідно до порівняння координат джерел нектару. Коли джерело нектару залишається бджолами, бджола-розвідник має визначити нове джерело довільним чином і бджоли мають переключитися на нього.

В даному алгоритмі координати кожного з джерел нектару є потенційними розв'язками задачі оптимізації, а кількість нектару в джерелах відповідає фітнес-функції (або якісній характеристиці) відповідного з розв'язків. Кількість потенційних координат є рівною кількості робочих бджіл та спостерігачів.

На першому етапі ABC алгоритм генерує початкову популяцію, що є довільно розподілена для SN розв'язків (координати джерел нектару). Кожен з розв'язків x_i $(i = 1,2,\dots,SN)$ являє собою D-вимірний вектором. Колонія після ініціалізації зазнає змін впродовж ітерацій $C = 1,2,\dots,MCN$, під час процесів пошуку робочими бджолами, бджолами-спостерігачами та бджолами-розвідниками. Робочі бджоли змінюють свої координати в пам'яті залежно від локальної інформації (візуальної інформації) та досліджують кількість нектару нового джерела. У випадку, коли кількість нектару нового джерела є вищою за попередню, бджола запам'ятовує нові координати і видаляє з пам'яті попередні. В іншому випадку бджола залишає пам'ять в такому ж стані. Після завершення процесу пошуку усіма

робочими бджолами, вони поширюють інформацію щодо нектару в джерелах та координати джерел зі спостерігачами на майданчику для танців. Спостерігачі виконують оцінку інформації щодо нектару, яка отримана від усіх робочих бджіл та виконують вибір джерела нектару з ймовірністю, яка залежить від кількості нектару. Бджола-спостерігач змінює в своїй пам'яті інформацію про джерело нектару та оцінює кількість нектару в потенційному джерелі таким же чином, як це виконують робочі бджоли. У випадку, коли нове джерело містить більше нектару за попереднє, бджола розташування цього нового джерела та видаляє з пам'яті інформацію про попереднє [3].

1.2.2. Мурашиний алгоритм

Мураха в ACO – це стохастична конструктивна процедура, яка поступово знаходить розв'язок шляхом додавання короткочасно визначених компонентів розв'язку до часткового розв'язку, що обчислюється. Тому метаевристика ACO може застосовуватися до будь-якої комбінаторної задачі оптимізації, для якої може бути визначена конструктивна евристика. Хоча це означає, що метаевристика ACO може бути застосована до будь-яких комбінаторних задач оптимізації, проблема полягає в тому, як використати розглянуту проблему до форми, яка може використовуватися мурахами для побудови розв'язків.

Як ми вже говорили, в алгоритмах ACO мурашки є стохастичними конструктивними процедурами, які будують розв'язок, переміщаючись по графу $G_C = (C, L)$, де множина L повністю з'єднує елементи C. Обмеження задачі $\Omega(t)$ вбудовані в евристики мурах. У більшості випадків мурашки отримують допустимі розв'язки. Проте іноді може бути корисно також дозволити їм будувати недопустимі розв'язки. Вершини $c_i \in C$ і ребра $l_{ij} \in$

L мають відповідні *сліди феромонів (pheromone trail)* τ (сліди τ_i пов'язані з вершинами, слід τ_{ij} пов'язані з ребрами), а також *евристичне значення* η (η_i і η_{ij} відповідно). Слід феромонів заносить в довготривалу пам'ять інформацію про весь процес пошуку мурашки та оновлюється самими мурахами. Інакше кажучи, евристичне значення, яке часто називають евристичною інформацією, являє собою апріорну інформацію про поточну задачу або інформацію, що отримується в момент виконання, що надається джерелом, відмінним від мурашок. У багатьох випадках η – це вартість або оцінка вартості додавання елементів до розробленого розв'язку. Ці значення використовуються евристичними правилами мурах, щоб зробити ймовірнісні висновки про те, як переміщатись по графу.

Точніше, кожна мурашка колонії має наступні властивості:

- Використовує граф $G_C = (C, L)$ для пошуку найкоротших розв'язків $s^* \in S^*$;
- Має пам'ять M^k, яку може використовувати для зберігання інформації про шлях, яким вона пересувалася раніше. Пам'ять може бути використана для (1) побудови допустимих розв'язків (тобто реалізації обмежень Ω); (2) обчислень евристичних значень; (3) оцінки знайдених розв'язків; (4) відслідковування зворотного шляху;
- Має початковий стан x_k^s і одну або більше *умову зупинки* e^k. Зазвичай початковий стан виражається або як порожня послідовність, або як послідовність одиничної довжини, тобто послідовність з одним елементом;
- Коли мураха знаходиться в стані $x_r = \langle x_{r-1}, i \rangle$, якщо умова зупинки не виконується, вона рухається до сусідньої вершини j, тобто до стану $\langle x_r, j \rangle \in \chi$. Якщо принаймні одна із умов зупинки e^k виконується, то мурашка зупиняється. Коли мураха обчислює потенційний розв'язок,

переміщення в недопустимий стан забороняється в більшості програм, як за допомогою пам’яті мурашки, так і за допомогою відповідних евристичних значень η;

- Виконує переміщення застосовуючи правило ймовірнісного вибору. Правило ймовірнісного вибору є: (1) функцією локально доступних феромонів і евристичних значень (тобто феромонів і евристичних значень, пов’язаних з вершинами та ребрами в околиці поточного розташування мурашки на графі G_C); (2) приватною пам’яттю мурашки, що зберігає її поточний стан; (3) обмеженнями задачі;
- При додаванні вершин c_j до поточного стану, вона може оновити слід феромонів, пов’язаний з нею або з відповідним ребрами;
- Після того, як мураха знайшла розв’язки, вона може пройти той самий шлях у зворотному напрямку і оновити сліди феромонів для вершин.

Важливо зазначити, що мурашки діють всі разом і незалежно одна від одної одночасно, і хоча кожна мураха є достатньо складною, щоб знайти (хоч і не дуже якісний) розв’язок поставленої задачі, якісні розв’язки можуть бути отримані тільки в результаті колективної взаємодії між мурахами. Це досягається через непряму комунікацію, опосередковану інформаційними мурахами, які читають чи записують дані до змінних, що зберігають значення сліду феромонів. У певному сенсі це – розподілений процес навчання, в якому окремі агенти, мурашки, не тільки є адаптивними, а, навпаки, адаптують спосіб, в який представлена здача та сприймається іншими мурахами [4].

1.2.3. Алгоритм зозулі

Алгоритм пошуку зозулі базується на природному паразитизмі деяких видів зозулі шляхом закладання яєць у гнізда птахів-господарів. Ці зозулі-

паразити жіночого роду можуть імітувати кольори та візерунок яєць птахів-господарів. Для простоти, передбачається, що в гнізді існує лише одне яйце. Наявне яйце в гнізді являє собою початковий розв'язок. Яйце, закладене зозулею, являє собою новий розв'язок, породжене алгоритмом. Алгоритм працює на основі трьох припущень:

1. Зозуля вибирає гніздо, щоб закласти яйце випадковим чином, і закладає лише одне яйце за раз;
2. Найкращі гнізда з якісними яйцями (розв'язками) переносяться на наступні покоління;

Загальна кількість наявних гнізд для господарських птахів є фіксованою, і птах-господар може виявити яйце зозулі з ймовірністю $P_\alpha \subset [0;1]$. У цьому випадку птах-господар або знищує яйце, або повністю знищує гніздо, і збирає нове гніздо в іншому місці. Третє припущення можна наблизити до частини P_α усіх n гнізд, які замінюються новими гніздами, що мають новий випадковий розв'язок. Коли створюється новий розв'язок $X^{(t+1)}$ для, скажімо, зозулі i, Levy Flight виконується як $X_i^{(t+1)} = X_i^{(t)} + \alpha \oplus Levy(\lambda)$, де $\alpha > 0$ – розмір кроку, який повинен бути пов'язаний із масштабами задачі. У більшості випадків використовується $\alpha = 1$. Дане рівняння є стохастичним рівнянням довільного переміщення. Це довільне переміщення є ланцюгом Маркова, коли наступне розташування залежить лише від поточного розташування та ймовірності переходу. $\oplus$ означає попарний добуток елементів. Levy flight по суті забезпечує довільне переміщення, тоді як довільна довжина кроку складається з розподілу $Levy{\sim}u = t^{-\lambda}, (1 < \lambda \leq 3)$, що має нескінченну дисперсію з нескінченним середнім значенням. Тут кроки по суті утворюють процес довільного переміщення з розподілом довжини кроку з хвостом, що повільно

зменшується. Алгоритм також може бути розширений на більш складні випадки, коли кожне гніздо містить кілька яєць (набір розв'язків) [5].

1.2.4. Алгоритм кажанів

Якщо ми ідеалізуємо деякі ехолокаційні характеристики кажанів, ми можемо розвивати різні алгоритми, в основі яких знаходиться їх поведінка. Для простоти, ми будемо використовувати наступні ідеалізовані правила:

1. Всі кажани використовують ехолокацію для вимірювання дистанції і вони також «знають» різницю між їжею/здобиччю та фоновими бар'єрами в якийсь магічний спосіб;
2. Кажани літають випадково зі швидкістю v_i, положенням x_i з фіксованою частотою f_{min} та змінними довжиною хвилі λ і гучністю A_0 для пошуку здобичі. Вони можуть автоматично коригувати довжину хвилі (або частоту) своїх імпульсивних випромінювань та регулювати частоту імпульсного випромінювання $r \in [0,1]$ залежно від відстані до цілі;
3. Незважаючи на те, що гучність може змінюватись багатьма способами, ми припускаємо, що вона змінюється від великого (додатного) A_0 до мінімального постійного значення A_{min}.

Для нашої задачі ми можемо використовувати будь-яку довжину хвилі для зручності. У фактичній реалізації ми можемо налаштувати діапазон шляхом регулювання довжин хвиль (або частоти), а також діапазон, який можна виявити (або найбільшу довжину хвилі) такий, що на початку має величину як і розмір області пошуку, а потім зменшується до менших значень. Крім того, ми не обов'язково повинні використовувати довжини хвиль, замість цього ми можемо змінювати частоту при фіксованій довжині

хвилі λ. Це можливо завдяки тому, що λ та f пов'язані залежністю, і значення λf буде сталим. Ми будемо використовувати цей підхід в нашій реалізації.

Для простоти можна вважати, що $f \in [0, f_{max}]$. Ми знаємо, що вищі частоти мають коротші довжини хвиль і сягають коротших відстаней. Для кажанів типові відстані – це кілька метрів. Величина імпульсу може бути в діапазоні $[0, 1]$, де 0 означає відсутність імпульсів, а 1 – максимальна величина їх випромінення [6].

1.2.5. Алгоритм світлячків

Алгоритм світлячків (Firefly algorithm, FA) – це метаевристичний алгоритм, розроблений доктором Синь-Ши Яном (англ. Dr. Xin-Shi Yang). Цей алгоритм базується на природній поведінці світлячків, яка базується на явищі біолюмінесценції. Світлячки в природі здатні виробляти світло завдяки спеціальним фотогенним органам, розташованим дуже близько до поверхні тіла за оболонкою напівпрозорої кутикули. Це світло допомагає їм спілкуватися одне з одним і приваблювати інших світлячків.

Поведінка світлячків

Процес біолюмінесценції відповідає за випромінення світла світлячками. Є багато теорій, що стосуються причини та важливості миготливого світла в життєвому циклі світлячка, але більшість з них пояснюють це фазою спарювання. Основне завдання миготливого світла полягає в тому, щоб привабити партнера. Шаблони цих ритмічних спалахів унікальні на основі швидкості миготіння та часу, протягом якого спостерігаються спалахи. Цей шаблон приваблює як особин чоловічого роду, так і жіночого одне до одного. Згідно зі зворотним квадратичним законом, інтенсивність світла I зменшується при зростанні відстані r.

Повітря також поглинає світло, і воно стає слабшим із збільшенням відстані. Поєднання цих двох чинників зменшує видимість світлячків на обмежену відстань, яка, як правило, становить кількасот метрів уночі, проте є достатньою для того, щоб світлячки могли спілкуватися один з одним.

Принцип

Алгоритм світлячків використовує дану біологічну інформацію, сформовану в три ідеалізовані правила:

1. Усі світлячки є безстатевими. Тобто один світлячок взаємодіє з іншими незалежно від статі.

2. Привабливість світлячка пропорційна його яскравості. Таким чином, у випадку двох світлячків, менш яскравий рухається в напрямку більш яскравого. Оскільки привабливість пропорційна яскравості, вона зменшується зі збільшенням відстані між ними.

3. Значення функції, що оптимізується обернено пропорційне яскравості світлячка.

Для задачі максимізації вважається, що яскравість світлячків має бути пропорційною фітнес функції. Інші форми яскравості можна визначити так само як функцію фітнесу на зразок генетичних алгоритмів.

Інтенсивність світла та привабливість

Алгоритм світлячків ґрунтується на двох важливих речах: варіації інтенсивності світла та формулюванні поняття привабливості. Для простоти передбачається, що привабливість світлячка визначається її яскравістю, яка пов’язана з цільовою функцією.

У конкретному розташуванні x, яскравість I світлячка може бути обрана як $I(x)$ чи $\mathrm{f}(x)$ для задачі максимізації.

Привабливість β є відносною, що означає, що вона має оцінюватися іншими світляками, тому вона буде відрізнятися від відстані r_{ij} між

світлячком i та світлячком j. Як вже було зазначено, інтенсивність світла зменшується з відстанню від джерела, а світло також поглинається повітрям, тому привабливість повинна дозволяти змінюватися з різним ступенем поглинання. Отже, в найпростішому вигляді інтенсивність світла $I(r)$ змінюється в залежності від зворотного закону квадратів:

$$I(r) = \frac{I_S}{r^2}, \tag{2}$$

де I_S – яскравість світла джерела випромінювання.

Формулу (2) з урахуванням коефіцієнта поглинання можна подати у Гаусовій формі

$$I(r) = I_0 e^{-\gamma r}, \tag{3}$$

де I_0 – яскравість світла джерела випромінювання, γ – фіксований коефіцієнта поглинання світла (даний параметр можна подавати як залежний від відстані). Щоб уникнути сингулярності при $r = 0$ у виразі I_S/r^2, сукупний ефект як зворотного квадратного закону, так і поглинання можна апроксимувати як наступну Гаусову форму

$$I(r) = I_0 e^{-\gamma r^2}. \tag{4}$$

Як згадано раніше, привабливість світлячків β залежно від відстані між ними пропорційна яскравості світла I і описується аналогічною, до (4) формулою

$$\beta(r) = \beta_0 e^{-\gamma r^2}, \tag{5}$$

де β_0 – яскравість світла джерела випромінювання, γ – фіксований коефіцієнта поглинання світла.

Оскільки часто буває швидше обчислити $1/(1 + r^2)$, ніж експоненціальну функцію, приведену вище, то (5) можна апроксимувати як:

$$\beta(r) = \frac{\beta_0}{(1 + \gamma r^2)}. \tag{6}$$

Обидва рівняння (5) і (6) визначають характерну відстань $1/\gamma$, на якій привабливість істотно змінюється від β_0 до ${\beta_0}^{-1}$ для (5) або $\beta_0/2$ для (6). У реалізації для реального часу, $\beta(r)$ – це функція привабливості, яка може бути будь-якою функцією, що монотонно зменшується, наприклад:

$$\beta(r) = \beta_0 e^{-\gamma r^m} \ (m \geq 1). \tag{7}$$

При фіксації, характеристична довжина набуває вигляду:

$$\Gamma = \gamma^{-\frac{1}{m}} \to 1, m \to \infty. \tag{8}$$

І навпаки, параметр γ може бути використаний як типове початкове значення для певної довжини шкали Г задачі оптимізації. Наприклад:

$$\gamma = \frac{1}{\Gamma^m}. \tag{9}$$

Відстань $r_{i,j}$ між світлячками i та j обчислюється за формулою евклідової відстані

$$r_{i,j} = \|x_i - x_j\| = \sqrt{\sum_{k=1}^{d} \left(x_{i,k} - x_{j,k}\right)^2}, \tag{10}$$

де d – вимірність простору.

Для двовимірного простору формула (10) набуває вигляду

$$r_{i,j} = \sqrt{\left(x_i - x_j\right)^2 - \left(y_i - y_j\right)^2}, \tag{11}$$

З використанням формул (3) і (5) отримуємо формулу переміщення світлячка i до світлячка j

$$x_{i,k}^{t+1} = x_{i,k}^t + \beta_0 e^{-\gamma r_{i,j}^2}\left(x_{j,k}^t - x_{i,k}^t\right) + \alpha €_i, \tag{12}$$

де k – номер координати, $x_{i,k}^t$ – значення координати до переміщення, α_t – довільний коефіцієнт, $€_i$ – i-й елемент вектора довільних чисел.

Другий компонент використовується для привабливості, а третій – для рандомізації, причому α є параметром рандомізації, а $€_i$ – вектор

випадкових чисел, який отримується з Гаусового або рівномірного розподілу. Наприклад, найпростіший форма – коли $€_i$ можна замінити на $(rand - 0{,}5)$, де $rand$ – генератор випадкових чисел, розподілених у рівномірному діапазоні [0, 1]. Найважливішим параметром у алгоритмі світлячків є γ, він відіграє дуже важливу роль у визначенні швидкості збіжності та поведінки алгоритму FA. Теоретично $\gamma \in [0, \infty)$, але на практиці $\gamma = O(1)$ визначається характеристичною довжиною Г оптимізованої системи. Так що для майже всіх програм він змінюється від 0,1 до 10.

Якщо $\beta_0 = 0$, це стає простим випадковим переміщенням. З іншого боку, якщо $\gamma = 0$, алгоритм зводиться до варіанту PSO. Крім того, рандомізація $€_i$ може бути легко поширена на інші розподіли, такі як польоти Леві.

Оскільки α_t істотно контролює випадковість (або, в якійсь мірі, різноманітність розв'язків), ми можемо налаштувати цей параметр під час ітерацій так, щоб він міг змінюватися залежно від лічильника ітерацій t. Таким чином, α_t можна представити як

$$\alpha_t = \alpha_0 \delta^t, 0 < \delta < 1,$$

де α_0 – початковий коефіцієнт масштабування, який є випадковим, і є по суті фактором сповільнення. Для більшості програм ми можемо використовувати δ від 0,95 до 0,97.

Що стосується початкового α_0, то моделювання показують, що FA буде більш ефективним, якщо цей параметр пов'язаний з масштабуванням конструктивних змінних. Нехай L – середня значення досліджуваної функції. Тоді ми можемо встановити $\alpha_0 = 0{,}01L$. Множник 0,01 випливає з того факту, що для випадкових переміщень потрібно декілька кроків для

досягнення мети, одночасно врівноважуючи локальний пошук, не перестрибуючи більше ніж на кілька кроків.

Параметр β керує привабливістю, а параметричні дослідження показують, що $\beta_0 = 1$ може використовуватися для більшості програм. Однак, γ має також залежати від масштабування L. Загалом, ми можемо встановити $\gamma = 1/\sqrt{L}$. Якщо варіанти масштабування не є значними, то ми можемо встановити $\gamma = O(1)$ [7].

Алгоритм світлячків можна подати у вигляді наступного псевдокоду:

Цільова функція $f(x)$, $x = (x_1, x_2, \dots, x_d)^T$
Виконати ініціалізацію популяції світлячків x_i, $i = 1,2, \dots, n$
Задати коефіцієнт поглинання світла γ
WHILE ($t <$ ⟨максимальна кількість ітерацій⟩)
    FOR i=1: n (всі світлячки)
        FOR j=1: i
            Інтенсивність світла I_i в x_i визначається згідно $f(x_i)$
            IF ($I_i > I_j$)
                Перемістити світлячка i до світлячка j по всім координатам
            ELSE
                Перемістити світлячка i довільно
            END IF
            Обчислити нове значення цільової функції світлячка i
        END FOR
    END FOR
    Обчислити ранг світлячків згідно інтенсивності світла та визначити найкращого світлячка
END WHILE

---

2. РОЗРОБКА МЕТОДУ РОЮ ЧАСТИНОК ПОШУКУ НАЙКОРОТШОГО ШЛЯХУ В МЕРЕЖІ

2.1. Оригінальний метод рою частинок

Задачі комбінаторної оптимізації інтригують, оскільки можна легко описати постановку задачі, але дуже важко їх вирішити. Багато проблем, що виникають у програмах, є NP-складними задачами, тобто вважається, що вони не можуть бути оптимально розв’язані в межах поліноміально обмеженого часу. Отже, щоб практично вирішувати такі випадки, часто доводиться використовувати приблизні методи, які повертають розв’язки, близькі до найкоротших за відносно короткий час. Алгоритми цього типу називають евристичними. Вони часто використовують певні специфічні знання, щоб знаходити або вдосконалювати розв’язок.

Останнім часом багато дослідників зосереджують свою увагу на новому класі алгоритмів, що називаються метаевристичними. Метаевристика – це набір алгоритмічних понять, які можуть бути використані для визначення евристичних методів, що застосовуються для широкого кола різних завдань. Використання метаевристики значно підвищує можливість знаходження якісного розв’язку для складних, актуальних комбінаторних задач оптимізації за розумний час.

Інакше кажучи, метаевристичний метод може розглядатися як загальноприйнятий евристичний метод, призначений для розв’язання конкретної евристичної задачі (наприклад, алгоритму локального пошуку) і може бути використаний для інших умов (наприклад, дослідження перспективних областей пошукового простору, що містять якісні розв’язки). Отже, метаевристика є загальною алгоритмічною обгорткою, яка може бути застосована до різних задач оптимізації з відносно невеликою кількістю модифікацій, щоб зробити її адаптованою до конкретної проблеми.

Метаевристика – досить невдала назва для опису великого підрозділу, в дійсності основного, в стохастичній оптимізації. Стохастична оптимізація є великим класом алгоритмів і методів, які в тій чи іншій мірі використовують випадковість для пошуку оптимального (або досяжного оптимального) розв'язку складних завдань. Метаевристики — найзагальніші алгоритми з цього класу і застосовуються для розв'язання широкого спектру завдань.

Метаевристики застосовуються до завдань типу «я впізнаю їх, коли побачу». Це алгоритми, які використовуються для пошуку розв'язку на завдання, в яких доступно дуже мало допоміжної інформації: ви не знаєте, на що схожий оптимальний розв'язок, невідомо, яким способом необхідно шукати цей розв'язок, наявної евристичної інформації явно недостатньо, а послідовний перебір не підходить, тому що простір пошуку дуже великий. Але якщо є потенційний розв'язок, то його можна перевірити і встановити, наскільки воно добре. Тобто можна дізнатися якість розв'язку, коли воно доступно.

Припустимо, наприклад, що необхідно знайти оптимальний набір дій для робота, який грає у футбол. Є симулятор дій робота і можна протестувати будь-який набір дій і присвоїти цьому набору оцінку (хороший набір дій можна побачити). Також є загальне визначення для набору дій, тобто можна уявити, що він із себе представляє. Однак немає жодної ідеї про те, яким повинен бути оптимальний набір, і про те, як до нього прийти [9].

Найпростіший варіант розв'язання задачі в подібній ситуації — випадковий пошук: будемо просто генерувати випадкові набори поводжень, поки не закінчиться відведений на пошук час, і повернемо найкращий знайдений набір. Але перш ніж впасти у відчай і взятися за випадковий

пошук, розглянемо альтернативу, таку як локальний пошук. Почнемо працювати з випадковим набором дій. Потім застосуємо до нього невелику випадкову зміну і протестуємо отриману нову версію. Якщо вона виявиться краще, то попередню версію відкинемо. В іншому випадку, відкидається нова версія. Після зробимо ще одне невелику випадкову зміна поточної версії набору (тобто тієї, яка не була відкинута). Якщо нова версія краща, то відкидаємо поточну, інакше не приймаємо нову. Повторюємо процес наскільки це можливо.

Локальний пошук – це простий метаевристичний алгоритм. Він використовує евристичне припущення про властивість простору пошуку, яке зазвичай актуально для багатьох завдань: схожі розв'язки часто ведуть себе подібним чином (і часто мають близьку якість), тому невеликі модифікації розв'язку зазвичай тягнуть невеликі і зрозумілі зміни в їхній якості, що дозволяє підніматися на «вершину гори якості» для отримання якісного розв'язку. Таке евристичне припущення є однією з ключових особливостей метаевристичних методів. І дійсно, практично всі метаевристики об'єднують локальний пошук з випадковим.

Метаевристики в даній роботі цікавитимуть нас саме у розрізі проблеми локального пошуку.

Опишемо математичну структуру. Нехай $A \subset R^n$ – простір пошуку, а $f: A \to Y \subset R$ – цільова функція. Для максимально простого опису, ми припускаємо, що A є також можливим розв'язком функції, тобто не існує жодних явних обмежень на потенційні розв'язки. Зауважимо, що немає необхідності в додаткових припущеннях щодо форми цільової функції та простору пошуку. PSO – це алгоритм на основі популяції, тобто він експлуатує популяцію потенційних розв'язків для одночасного точкового пошуку в просторі. Популяція називається *роєм*, його особи – *частками*, а нотації використовується як і для подібних моделей у соціальних науках та

фізиці частинок. Рій визначається як множина $S = \{x_1, x_2, \dots, x_N\}$ з N частинок (потенційні розв'язки), кожна з яких має вигляд $x_i = \left(x_{i_1}, x_{i_2}, \dots, x_{i_n}\right)^T \subset A, i = 1, 2, \dots, N$ [10].

Індекси довільно визначаються для частинок, а N – визначений користувачем параметр алгоритму. Цільова функція $f(x)$ вважається доступною для всіх точок A. Отже, кожна частка має унікальну функціональну величину $f_i = f(x_i) \subset Y$.

Передбачається, що частинки будуть рухатися в пошуковому просторі A ітераційно. Це досягається шляхом регулювання їх положення за допомогою правильного зсуву розміщення, що називається швидкістю, і позначається як $v_i = \left(v_{i_1}, v_{i_2}, \dots, v_{i_n}\right)^T \subset A, i = 1, 2, \dots, N$.

Швидкість також адаптується ітераційно, щоб надати часткам здатності відвідувати будь-яку область в A. Якщо t – номер ітерації, то поточне положення частинки i та її швидкість буде надалі позначатися як $x_i(t)$ і $v_i(t)$ відповідно.

Швидкість оновлюється на основі інформації, отриманої на попередніх кроках алгоритму. Це реалізується в пам'яті, де кожна частка може зберігати найкращі координати, в яких вона коли-небудь була за час пошуку. Для цього, крім рою S, який містить поточні розміщення частинок, PSO містить також набір пам'яті $P = \{p_1, p_2, \dots, p_N\}$, який містить найкращі координати $p_i = \left(p_{i_1}, p_{i_2}, \dots, p_{i_n}\right)^T \subset A, i = 1, 2, \dots, N$, коли-небудь відвідані кожною часткою. Ці розміщення позначаються як: $p_i(t) = f_i(t)$, де t – номер ітерації.

Алгоритм PSO оснований на імітації моделі соціальної поведінки. Таким чином, існує механізм обміну інформацією, щоб частинки могли взаємно повідомляти свій досвід. Алгоритм апроксимує глобальний мінімум за найкращими координатами, які коли-небудь відвідали всі

частинки. Тому це розумний вибір для обміну даною ключовою інформацією. Нехай g – індекс найкращих координат з найменшим значенням функції в P на ітерації t, тобто $p_i(t) = argmin\big(f_i(t)\big)$.

Тоді базову версію PSO можна описати наступними рівняннями:

$$\begin{aligned} v_{ij}(t+1) &= v_{ij}(t) + c_1 R_1 \left(p_{ij}(t) - x_{ij}(t)\right) + c_2 R_2 \left(p_{gj}(t) - x_{ij}(t)\right) \\ x_{ij}(t+1) &= x_{ij}(t) + v_{ij}(t+1) \end{aligned} \quad (1)$$

Псевдокод алгоритму PSO:

На вході: кількість частинок (N); рій (S); найкращі координати (P)

Крок 1. Встановити $t \leftarrow 0$

Крок 2. Ініціалізувати S і встановити $P = S$

Крок 3. Оцінити S та P, і визначити індекс g *найкращих координат*

Крок 4. While (критерій зупинки не виконується)

Крок 5. Оновити S за допомогою рівнянь (1) та (2)

Крок 6. Обчислити S

Крок 7. Оновити P та перевизначити індекс g

Крок 8. Встановити $t \leftarrow t + 1$

Крок 9. End While

Крок 10. Визначити найкращі координати

Тут $i = 1, 2, \dots, N, j = 1, 2, \dots, n, t$ – номер ітерації; R_1 і R_2 – випадкові величини, рівномірно розподілені в межах $[0, 1]$; c_1, c_2 є ваговими факторами, які також називаються когнітивним та соціальним параметрами відповідно [11].

На кожній ітерації після оновлення та оцінки частинок оновлюються найкращі розміщення (пам’ять). Таким чином, нове найкраще розміщення x_i на ітерації $t + 1$ визначається наступним чином

$$p_i(t+1) = \begin{cases} x_i(t+1), \text{якщо } f\big(x_i(t+1)\big) \le f\big(p_i(t)\big) \\ p_i(t), \text{інакше} \end{cases}.$$

Нове визначення показника g для оновлених координат завершує ітерацію PSO. Частинки, як правило, ініціалізуються довільним чином, згідно рівномірного розподілу в пошуковому просторі A. Цей варіант виконує однаковий пошук у кожній з областей A; тому в основному це краще в тих випадках, коли немає інформації про форму пошукового простору або цільову функцію, що потребує іншої схеми ініціалізації. Крім того, він реалізується досить легко, оскільки всі сучасні комп'ютерні системи можуть бути оснащені єдиним генератором випадкових чисел.

Попереднє значення швидкості $v_{ij}(t)$ у правій частині рівняння (1), забезпечує засіб інерційного руху до частинки, беручи до уваги своє попереднє положення. Ця властивість може запобігти тому, що частка «захопилася» б пошуком найкращої позиції і могла б потрапити до локального мінімуму.

Крім того, попереднє значення швидкості служить збудженням для найкращої частинки x_g. Дійсно, якщо частка x_i виявляє нову позицію з нижчим значенням функції, ніж найкраще значення, то вона стає найкращою у рої (тобто $g \leftarrow i$), а її найкраща позиція p_i, буде збігатися з p_g та x_i на наступній ітерації. Таким чином, дві стохастичні частини в рівнянні (1) зникнуть. Якщо в рівнянні (1) не було попереднього значення швидкості, то вищезгадана частка залишатиметься в одному і тому ж положенні протягом кількох ітерацій, допоки нове найкраще положення не буде виявлено іншою часткою. На відміну від цього, доданок швидкості дозволяє цій частині продовжувати пошук в напрямі його колишнього зсуву розміщення.

Значення c_1 і c_2 можуть впливати на пошукову спроможність PSO шляхом зміщення напряму вибору нових позицій частинки x_i, відповідно до кращих позицій x_i і p_g.

Очевидно що величина пошуку значно відрізняється у двох випадках. Якщо потрібне краще глобальне дослідження, тоді більші значення c_1 і c_2 можуть забезпечити нові точки у відносно далеких областях пошукового простору. З іншого боку, більш детальний локальний пошук за найкращими позиціями, досягнутими до теперішнього часу, потребує вибору менших значень для двох параметрів. Крім того, вибравши, $c_1 > c_2$, буде відбуватися рух частинок у напряму p_i, тоді як у протилежному випадку, $c_1 < c_2$ буде направляти рух частинок у напрямку p_g. Цей ефект може бути корисним у випадках, коли існує інформація щодо форми цільової функції. Наприклад, в опуклих унімодальних цільових функціях вибір, який сприяє руху частинок у напрямку p_g, очікується, буде більш ефективним, якщо його поєднувати з правильною пошуковою величиною.

Слід зауважити, що PSO діє на кожному напрямку координат самостійно. На цьому етапі ми згадаємо типову помилку, зроблену кількома дослідниками, особливо, коли в їхній векторній формі розглядаються рівняння PSO:

$$v_i(t+1) = v_i(t) + c_1 R_1 \left(p_i(t) - x_i(t)\right) + c_2 R_2 \left(p_g(t) - x_i(t)\right) \quad (3)$$

$$x_i(t+1) = x_i(t) + v_i(t+1) \quad (4)$$

тут $i = 1, 2, \ldots, N$, де всі векторні операції виконуються покординатно. У цьому випадку R_1 та R_2 слід розглядати як випадкові n-вимірні вектори з їх компонентами, рівномірно розподіленими в межах [0,1]. Замість цього R_1 і R_2 часто розглядаються як випадкові одновимірні значення, подібно до рівняння (1), в результаті чого виходить схема, яка використовує ту ж

довільну кількість для всіх компонентів напрямку відповідного різницевого вектора в рівнянні (3).

У більшості додатків для оптимізації бажано або неминуче розглянути лише частинки, що лежать в межах пошукового простору. Для цього границі накладаються на положення кожної частинки x_i, щоб обмежити її в пошуковому просторі A. Якщо частка, що виконує надто великий крок з пошукового простору після застосування рівняння (2), то вона негайно повертається на границю. У найпростішому випадку, де простір пошуку можна визначити як

$$A = [a_1, b_1] \times [a_2, b_2] \times \ldots \times [a_n, b_n], \quad (5)$$

З $a_i, b_i \in R$, $i = 1,2, \ldots, n$ частинки мають обмеження

$$x_{ij}(t+1) = \begin{cases} a_j, \text{якщо } x_{ij}(t+1) < a_j \\ b_j, \text{якщо } x_{ij}(t+1) > b_j \end{cases},$$

де $i = 1,2, \ldots, N$, $j = 1,2, \ldots, n$.

В іншому випадку, розглядається рух зі «стрибанням» з-за границі назад до пошукового простору, подібно до м’яча, що відбивається від стіни. Популярність цього підходу обмежена, оскільки він вимагає моделювання руху частинок з комплексними фізичними рівняннями. У тих випадках, коли A не можна визначити як простір кубічного вигляду, для обмеження частинок можуть знадобитися спеціальні умови, що залежать від функції для оптимізації [8].

2.2. Модифікації методу

Задача пошуку найкоротшого шляху (SPP) може визначатися наступним чином. Нехай в математичній моделі маємо неорієнтований граф (узагальнений тип мережі) $G = (V, E)$, що містить набір вузлів $V = \{v_i\}$ і відповідну сукупність ребер $E \in V \times \mathrm{V}$, що з'єднують множину вузлів V.

Відповідно до кожного ребра, є невід'ємний номер c_{ij}, що представляє собою, можна так сказати, вартість (відстань, час переходу по ребру тощо) ребра від вузла v_i до вузла v_j. Шлях від вузла v_i до вузла v_k – це не що інше як послідовність вузлів (v_i, v_j, v_l, ..., v_k), в яких ні один з вузлів не повторюється.

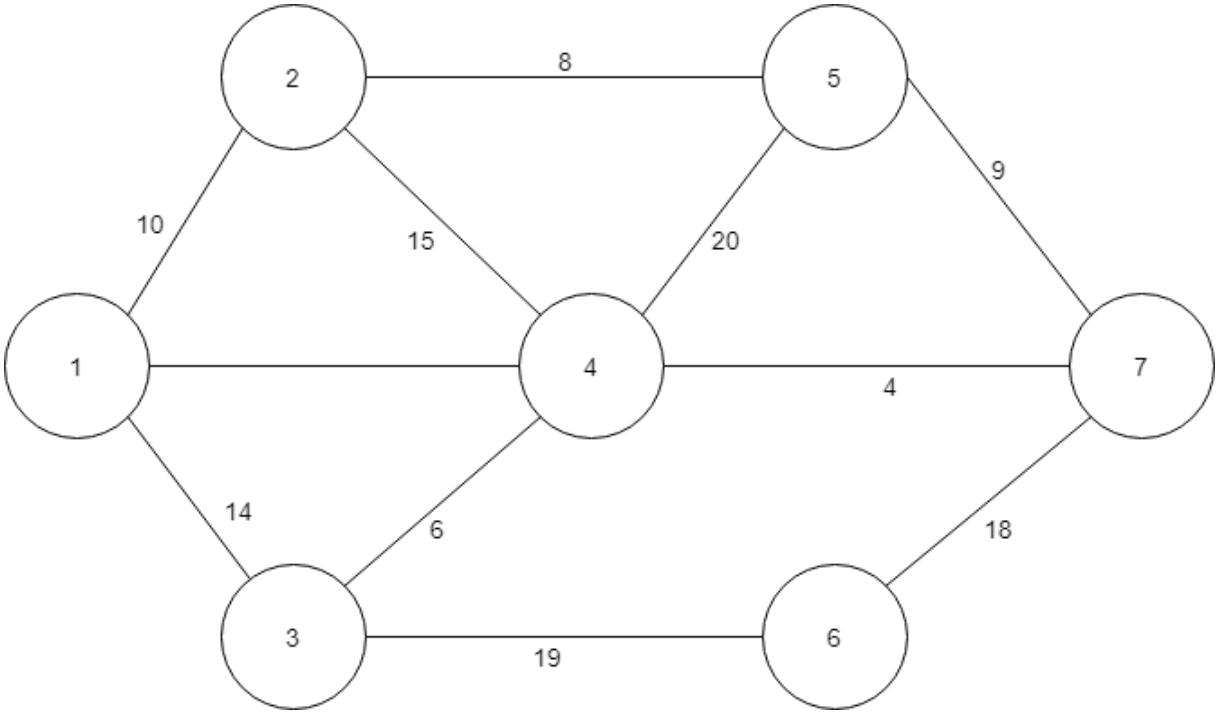


Рис. 2.2.1 – Приклад мережі

Наприклад, на шлях від вузла 1 до вузла 7 представлений як (1, 4, 2, 5, 7). Метою задачі пошуку найкоротшого шляху є наступне: знайти такий шлях між двома заданими вузлами, загальна вартість у сенсі суми ваг каналів зв'язку якого є мінімальною.

Лінійну програмну модель задачі пошуку найкоротшого шляху можна представити наступним чином (s і t – початкова і кінцева вершини відповідно).

Мінімізувати

$$\sum_{(i,j)\in E}^{h_{uj}} c_{ij},$$

так, щоб

$$\sum_{j:(i,j)\in E} h_{ij} - \sum_{j:(i,j)\in E} h_{ji} = \begin{cases} 1, i = s \\ -1, i = t, \\ 0, i \neq s, t \end{cases}$$

де h_{ij} – це 1, якщо ребро з'єднує вершини i та j на шляху та 0 інакше.

Запропонований гібридний алгоритм використовує псевдокоди PSO, перераховані в алгоритмі 2.1 для розрахунку найкоротших шляхів в телекомунікаційній мережі з включенням метаевристик, що мають відношення диверсифікований локальний пошук. У алгоритмі рою частинок якість частинки (розв'язку) вимірюється фітнес функцією або, як її ще називають, функцією пристосування. Для задачі пошуку найкоротшого шляху в телекомунікаційній мережі функція фітнесу очевидна, оскільки метою є пошук мінімального шляху за вартістю його ребер по всій довжині. Таким чином, пристосованість (або ж фітнес-функція) i-ї частинки визначається як

$$f_i = \sum_{j=1}^{N_i-1} c_{yz},$$

$$y = PP^i(j),$$

$$z = PP^i(j+1),$$

де PP^i – це набір послідовних ідентифікаторів вузлів для i-ї частинки, а $N_i = |PP_i|$ – це кількість вузлів, які складають шлях, представлений i-ю

частинкою, а c_{yz} – вартість (або вага) ребра, що з'єднує вузол телекомунікаційної мережі y з вузлом z. Таким чином, функція пристосованості приймає мінімальне значення в тому випадку, коли шлях є найкоротшим. Якщо шлях, представлений частинкою, виявляється помилковим, його фітнес призначається штрафним значенням, так що дані атрибути не будуть розглядатись іншими частинками при майбутньому пошуку.

Функція пристосованості (англ. fitness function) – це особливий тип цільової функції, який застосовується як порівняльний показник якості для підбивання підсумку того, наскільки близьким є заданий конструктивний (тобто той, що знаходиться на етапі отримання) розв'язок до досягнення поставленої задачі.

Зокрема, в галузях генетичного програмування та генетичних алгоритмів кожен конструктивний розв'язок, як правило, представляють як набір чисел (який називають хромосомою). Ідея полягає в том, щоб після кожного кола тестування або моделювання вилучати n найгірших конструктивних розв'язків, і отримати n нових із найкращих. Кожен конструктивний розв'язок, таким чином, потребує присудження порівняльного показника якості, що вказував би, наскільки близько цей розв'язок підійшло до того, щоб відповідати загальним умовам, і цей показник породжується застосуванням функції пристосованості до результатів тестування або моделювання, отриманих від цього розв'язку.

Незважаючи на те, що в кінцевому рахунку, розв'язок для алгоритму знаходить вже не людина, а комп'ютер, саме людина має аналітично розробити функцію пристосованості. Якщо її розроблено погано, то алгоритм або збігатиметься на неприйнятному розв'язку, або зазнаватиме труднощів у тому, щоб взагалі досягти збіжності.

Крім того, функція пристосованості мусить не лише тісно корелювати з метою задачі, вона також має обчислюватися швидко і просто. Швидкість виконання є дуже важливою, оскільки типовий генетичний алгоритм, для того, щоб отримати якісний результат для нетривіальної задачі, мусить виконувати велику кількість ітерацій [15].

$f_{best} \leftarrow \infty$

$X_{best} \leftarrow null$

Виконати довільну ініціалізацію X_i

Виконати довільну ініціалізацію V_i

Обчаслити значення функції пристосованості $f(X_i)$

    $B_i \leftarrow X_i$

    $B_i^n \leftarrow X_j$ // j – це індекс найкращої сусідньої частинки

$iteration\_count \leftarrow 0$

// max_$iteration$ – максимальна кількість ітерацій

while ($iteration\_count$ < max_$iteration$)

    для кожної частинки i

        знайти B_i^n таке, що $f(B_i^n) < f(X_j)$, $\forall j \epsilon$ {сусіди i}

        if $f(X_i) < f(B_i)$

            $B_i \leftarrow X_i$

        if $f(B_i) < f_{best}$

            $f_{best} \leftarrow f(B_i)$

            $X_{best} \leftarrow B_i$

        Оновити V_i

        Оновити X_i

        Обчислити $f(X_i)$

$iteration\_count \leftarrow iteration\_count + 1$

end while

return X_{best}

Рис 2.2.2 – Проста імплементація алгоритму рою частинок

Основною проблемою при застосуванні PSO (GA) до задачі пошуку найкоротшого шляху в телекомунікаційній мережі є кодування мережевого шляху у частинку в термінах алгоритму рою частинок (PSO) (аналог хромосом в генетичних алгоритмах). Це кодування впливає на ефективність процесу розв'язку/пошуку.

Зазвичай використовують два типових методи кодування для представлення шляху при розв'язанні задач пошуку найкоротших шляхів у графах чи то мережах за допомогою генетичних алгоритмів, проте ці підходи є слабо дослідженими у сучасній науці. Вони є прямими та непрямими за своїми представленнями. У схемі прямого представлення хромосома генетичного алгоритму (GA) кодується як послідовність ідентифікаційних номерів вузлів (ідентифікаторів вузлів), що знаходяться на шляху від вихідного вузла до кінцевого вузла. В цьому випадку для кодування використовують хромосому зі змінною довжиною, довжина якої дорівнює кількості вузлів для кодування списку ідентифікаторів (також можна використовувати індекси) вузлів від вихідного вузла до пункту призначення на базі топологічної бази даних мережі. В [11] використовують ще одну подібну (але дещо відмінну) хромосомну фіксовану довжину, тобто кожен ген в хромосомі представляє ідентифікатор вузла, який вибирається випадковим чином з набору вузлів, пов'язаних з вузлом, що відповідає номеру його місцезнаходження. Недоліком цих прямих підходів є те, що випадкова послідовність ідентифікаторів вузла може не відповідати

дійсному шляху (що закінчується на кінцевому вузлі без будь-якої циклу), збільшуючи кількість отриманих недійсних шляхів.

Непряма схема для схеми представлення хромосом була запропонована Геном (англ. Прізвище Gen) в алгоритмі [13], де замість ідентифікаторів вузлів, які безпосередньо з'являються на представленні шляху, для керівництва відомостями про вузли, що складають шлях, використовується позначення шляху.

Основна інформація, що використовується в [12], є пріоритетами різних вузлів у мережі. Під час ініціалізації генетичного алгоритму ці пріоритети призначаються випадковим чином. Шлях створюється за допомогою процедури послідовного додавання вузлів, починаючи з вузла джерела і закінчуючи у пункті призначення, ця процедура називається стратегією зростання шляху. На кожному кроці будівництва шляху з хромосоми, зазвичай існує кілька вузлів (потенційних шляхів) і вибирається той, що має найвищий пріоритет, тобто додається до шляху, і процес повторюється аж до досягнення цільового вузла. Для ефективного декодування використовується динамічна матриця суміжності, що зберігається в комп'ютерній (програмній) реалізації [12] і оновлюється після кожного вибору вузла, щоб вибраний вузол не став кандидатом для майбутнього вибору (тобто для уникнення циклів).

Одним з основних переваг цього кодування є те, що розмір хромосоми фіксується швидше ніж його зміна (як у прямому кодуванні), що полегшує застосування різних операторів як мутація і кросовер. Одним з недоліків є те, що хромосома є всього лише «опосередковано» закодованою: вона не містить важливу інформацію про характеристики мережі, такі як вартості її ребер (каналів зв'язку). Фактично це кодування є дуже схожим на кодування випадкових чисел, що використовується для графічного представлення дерева в генетичних алгоритмах.

Інший варіант непрямого кодування хромосоми називається кодуванням з вагою (англ. weighted encoding) [12]. Аналогічно до пріоритетного кодування, хромосома є вектором значень, що називаються вагами. Цей вектор використовується для зміни параметрів задачі, наприклад, вартості ребер. Перш за все, початкова задача тимчасово змінюється шляхом зміщення вагових параметрів. Потім до неї застосовують нееволюційне евристичне декодування, пов'язане із задачою, що насправді генерує розв'язок для модифікованої вище проблеми. Цей розв'язок в кінці трактується і оцінюється за оригінальною (до змін) задачею.

Згідно до цих схем кодування, в [14] представлена схема, яка має принцип кодування/декодування на основі кореляції витрат і розроблена таким чином, щоб вона відповідала частинкам рою задачі пошуку найкоротшого шляху. Слід зауважити, що пряме кодування не підходить для частинок, оскільки для оновлення частинок використовуються арифметичні операції. У запропонованій схемі кодування частинок ґрунтується на пріоритетах вузла, а декодування базується на процедурі росту шляху зі знову ж урахуванням пріоритетів вузлів, а також вартості ребер. Частинка містить вектор значень пріоритету вузла (довжина частинки дорівнює кількісті вузлів). Щоб побудувати шлях від частинки від початкового вузла (вузла 1) до кінцевого вузла (вузол n), ребра додаються в шлях послідовно. На кожному кроці наступний вузол (вузол j) вибирається з вузлів, що мають прямі зв'язки з поточним вузлом, таким чином, що накопичений загальний розв'язок, пріоритет наступного вузла (p_j) та відповідна вартість ребра є мінімальною, тобто

$$j = min\{c_{ij}p_j | (i,j) \in E\}, \tag{2.2.1}$$

$$p_j \in [-1.0,1.0].$$

Пункти цього алгоритму узагальнені в алгоритмі, що представлений на рисунку 2.2.3. Пріоритети вузла можуть бути від'ємними або додатніми дійсними числами в діапазоні $[-1.0,1.0]$. Параметри задачі (вартості ребер телекомунікаційної мережі) є в даному випадку частиною процедури декодування. На відміну від кодування з пріоритетами, де вузол додається до часткового шляху, який базується лише на його пріоритеті, в представленій процедурі вузол додається до шляху на основі мінімального добутку пріоритету наступного вузла на вартість ребра, що з'єднує поточний вузол з наступним, який буде вибраний. Експериментальні результати показують перевагу цієї процедури над пріоритетним кодуванням, коли вона реалізується в межах реалізації алгоритму рою частинок [14]. Пошукова система алгоритму рою частинок виконується для оптимального набору пріоритетних значень вузлів, які в своїй комбінації призводять до отримання до найкоротшого шляху в даній телекомунікаційній мережі.

// i – початковий вузол
// j – сусідній до i вузол
// n – кінцевий вузол
// 1 дорівнює початковому вузлу
// $A(i)$ – набір вузлів, що є сусідами вузла i
// $PATH(k)$ – частковий шлях на кроці декодування k
// p_j – відповідний пріоритет вузла j в частинці P (вектор X)
// N_∞ – задане велике число
$Particle\_Decoding(P)$
$i \leftarrow 1,$

$p_1 \leftarrow N_\infty$,
$k \leftarrow 0$,
$PATH(k) \leftarrow \{1\}$
$while\ \left(\{j \in A(i), p_j \neq N_\infty\} \neq \emptyset\right)$
    $k \leftarrow k + 1$
    $j \leftarrow \text{argmin}\{c_{ij} p_j | j \in A(i), p_j \neq N_\infty\}$
    $i \leftarrow j$
    $PATH(k) \leftarrow PATH(k) \cup \{i\}$
    $p_i \leftarrow N_\infty$
    $if\ i = n$
        $return\ PATH(k)$
$end\ while$

$return$ Неправильний шлях

Рис 2.2.2 – Псевдокод процедури вартісного пріоритетного декодування (англ. cost-priority-based decoding)

З метою покращення продуктивності алгоритму рою частинок є важливим застосувати для нього певні модифікації, що також використовуються для інших метаевристичних алгоритмів.

З метою покращення можливостей локального пошуку алгоритму в перспективних областях доцільно застосувати множник w інерції швидкості, який зменшується по мірі виконання ітерацій. За рахунок цього частинки набувають властивостей, які забезпечують більш ретельне дослідження перспективних точок простору пошуку, що були знайдені на попередніх ітераціях.

Інерцію швидкості можна обчислити за формулою

$$w_{iter} = w_{max} + (w_{max} - w_{min}) \frac{iter}{I},$$

де w_{max} та w_{min} – верхня та нижня границі інерції відповідно, вони є константами й визначаються емпірично залежно від функції, що оптимізується, $iter$ – номер поточної ітерації алгоритму, I – максимальна кількість ітерацій.

Тоді формула зміни швидкості набуває вигляду

$$v_{ij}(t+1) = w v_{ij}(t) + \varphi_p R_1 \left(p_{ij}(t) - x_{ij}(t)\right) + \varphi_g R_2 \left(p_{gj}(t) - x_{ij}(t)\right) \quad (2.2.2)$$

З урахуванням даної модифікації псевдокод алгоритму рою частинок можна представити в наступному вигляді:

Згенерувати початкову конфігурацію:
Виконати довільну ініціалізацію X_i
Виконати довільну ініціалізацію V_i
Обчислити значення функції пристосованості $f(X_i)$
    $P_i \leftarrow \infty$
    $G \leftarrow \infty$
$iteration\_count \leftarrow 0$
// max_$iteration$ – максимальна кількість ітерацій

while ($iteration\_count$ < max_$iteration$)
    для кожної частинки i
        Обчислити V_i із застосуванням методу інерції швидкості відповідно до (2.2.2)

Оновити X_i

Обчислити значення функції пристосування $f(X_i)$

знайти B_i^n таке, що $f(B_i^n) < f(X_j)$, $\forall j \epsilon$ {сусіди i}

if $f(X_i) < f(P_i)$

$P_i \leftarrow X_i$

if $f(P_i) < f(G)$

$G \leftarrow P_i$

Оновити V_i

$iteration\_count \leftarrow iteration\_count + 1$

end while

return G

Рис 2.2.3 – Псевдокод алгоритму рою частинок з застосуванням методу інерції швидкості

Ще однією модифікацією алгоритму рою частинок є застосування методу звуження множника.

Використання даного методу дозволяє більш точно скеровувати напрям руху частинки з використанням коефіцієнтів φ_p та φ_g. Новий коефіцієнт обчислюється за формулою

$$\chi = \begin{cases} 2\left(\left|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}\right|\right)^{-1}, \varphi = \varphi_p + \varphi_g > 4 \\ 1, \varphi = \varphi_p + \varphi_g \leq 4 \end{cases}.$$

Завдання використання коефіцієнта стиснення полягає у запобіганні переходу швидкості за межі допустимих значень, таким чином не має необхідності застосування такого поняття як «стискання швидкості» (англ. velocity clamping) затискання не потрібна. Але Еберхарт і Ші (англ. Eberhart and Shi) [16] повідомляють, що найкращі показники можуть бути досягнуті

за рахунок застосування і методу стискання швидкості, і методу звуження множника.

В такому випадку формула зміни швидкості набуває вигляду

$$v_{ij}(t+1) = \chi \Big[w v_{ij}(t) + \varphi_p R_1 \left(p_{ij}(t) - x_{ij}(t) \right) + \varphi_g R_2 \left(p_{gj}(t) - x_{ij}(t) \right) \Big] \quad (2.2.3)$$

З урахуванням даної модифікації псевдокод алгоритму рою частинок можна представити в наступному вигляді:

Згенерувати початкову конфігурацію:
Виконати довільну ініціалізацію X_i
Виконати довільну ініціалізацію V_i
Обчислити значення функції пристосованості $f(X_i)$
    $P_i \leftarrow \infty$
    $G \leftarrow \infty$
$iteration\_count \leftarrow 0$
// max$_iteration$ – максимальна кількість ітерацій

while ($iteration\_count <$ max$_iteration$)
    для кожної частинки i
        Обчислити V_i із застосуванням методу звуження множника відповідно до (2.2.3)
        Оновити X_i
        Обчислити значення функції пристосування $f(X_i)$
        if $f(X_i) < f(P_i)$
            $P_i \leftarrow X_i$

if $f(P_i) < f(G)$

$G \leftarrow P_i$

Оновити V_i

$iteration\_count \leftarrow iteration\_count + 1$

end while

return G

Рис 2.2.4 – Псевдокод алгоритму рою частинок з застосуванням методу звуження множника

Ще однією з проблем базового алгоритму рою частинок є його передчасна збіжність до локальних оптимумів.

В цьому випадку алгоритм не продовжує покращувати якість розв'язку після певної кількості ітерацій. В результаті рій зазнає стагнації після певної кількості ітерацій і може привести до розв'язку, що є далеким від найкращого. Щоб уникнути передчасної збіжності рою частинки повторно ініціалізуються з випадковою швидкістю, коли вона застряє в локальному мінімумі. В цьому випадку можна застосовувати кілька стратегій: довільне скидання швидкості довільної кількості довільних частинок, скидання швидкості на основі формули залежності від часу (номеру ітерації) тощо [20].

З урахуванням даної модифікації псевдокод алгоритму рою частинок можна представити в наступному вигляді:

Згенерувати початкову конфігурацію:

Виконати довільну ініціалізацію X_i

Виконати довільну ініціалізацію V_i

Обчислити значення функції пристосованості $f(X_i)$

```
        P_i ← ∞
        G ← ∞
iteration_count ← 0
// max_iteration – максимальна кількість ітерацій

while (iteration_count < max_iteration)
        для кожної частинки i
                Обчислити V_i
                Оновити X_i
                Обчислити значення функції пристосування f(X_i)
                if f(X_i) < f(P_i)
                        P_i ← X_i
                if f(P_i) < f(G)
                        G ← P_i
                Оновити V_i
                Виконати скидання швидкості
iteration_count ← iteration_count + 1
end while

return G
```

Рис 2.2.5 – Псевдокод алгоритму рою частинок з застосуванням методу скидання швидкості

Вище описано проблему передчасної збіжності алгоритму, що викликає зацікавлення багатьох дослідників області метаевристичних алгоритмів. З метою покращити пошукову здатність алгоритму, застосуємо дещо видозмінений метод, що має назву селективної регенерації частинок.

Даний метод працює для випадку, коли аналітично вирішено використовувати φ_g більшим за φ_p. В загальному метод полягає у визначенні «відстані» в термінах значення функції пристосованості між кожною частинкою рою та глобально найкращою частинкою. Серед тих частинок, «відстань» яких є меншою за аналітично задане значення u довільним чином вибирається d відсотків частинок, для яких виконується реініціалізація. Основною метою цього покращення є намагання допомогти якійсь кількості частинок вирватись із зони можливого локального мінімуму, до якого може потрапити найкраща частинка й потягти за собою інші. Оскільки було б розумно зважати на той факт, що найкраща частинка могла все ж таки не потрапити до локального мінімуму, а знаходиться в околі найкращого розв'язку, при визначенні нового розміщення d відсотків частинок для кожної з їх координат з певною ймовірністю c вибирається значення відповідної координати найкращої частинки, а з ймовірністю $(1 - c)$ вибирається довільне значення з області пошуку. Оскільки певна кількість частинок щойно вийшла з околу найкращої частинки, то було б доцільно на певний час змінити когнітивні параметри поведінки алгоритму, щоб дані частинки не повернулися миттєво до найкращої. Для цього будемо змінювати значення φ_p на $2\varphi_p$ (тобто подвоювати) для певної кількості ітерацій z (на певний час).

З урахуванням даної модифікації псевдокод алгоритму рою частинок можна представити в наступному вигляді:

Згенерувати початкову конфігурацію:
Виконати довільну ініціалізацію X_i
Виконати довільну ініціалізацію V_i
Обчислити значення функції пристосованості $f(X_i)$

$P_i \leftarrow \infty$

$G \leftarrow \infty$

$iteration\_count \leftarrow 0$

// max_$iteration$ – максимальна кількість ітерацій

while ($iteration\_count$ < max_$iteration$)

для кожної частинки i

Обчислити V_i

Оновити X_i

Обчислити значення функції пристосування $f(X_i)$

if $f(X_i) < f(P_i)$

$P_i \leftarrow X_i$

if $f(P_i) < f(G)$

$G \leftarrow P_i$

Оновити V_i

Застосувати метод селективної регенерації

$iteration\_count \leftarrow iteration\_count + 1$

end while

return G

Рис 2.2.6 – Псевдокод алгоритму рою частинок з застосуванням методу селективної регенерації

З урахуванням всіх описаних вище модифікацій можемо записати результуючий псевдокод модифікованого алгоритму рою частинок

Згенерувати початкову конфігурацію:

Виконати довільну ініціалізацію X_i

Виконати довільну ініціалізацію V_i

Обчислити значення функції пристосованості $f(X_i)$

$P_i \leftarrow \infty$

$G \leftarrow \infty$

$iteration\_count \leftarrow 0$

// max$_iteration$ – максимальна кількість ітерацій

while ($iteration\_count <$ max$_iteration$)

для кожної частинки i

Обчислити V_i із застосуванням методів інерції швидкості та звуження множника

Оновити X_i

Обчислити значення функції пристосування $f(X_i)$

if $f(X_i) < f(P_i)$

$P_i \leftarrow X_i$

if $f(P_i) < f(G)$

$G \leftarrow P_i$

Оновити V_i

Застосувати метод скидання швидкості

Застосувати метод селективної регенерації

$iteration\_count \leftarrow iteration\_count + 1$

end while

return G

Рис 2.2.7 – Псевдокод алгоритму рою частинок з застосуванням методу селективної регенерації

2.3. Застосування розробленого методу до задачі пошуку найкоротших шляхів

Як викладено в теоретичних відомостях та науковому дослідженні вище, алгоритм рою частинок орієнтований на знаходження оптимуму функції багатьох змінних, його застосування до пошуку шляхів у мережах не є очевидним на перший погляд.

Для застосування алгоритму рою частинок до задачі пошуку найкоротших шляхів у телекомунікаційній мережі, з врахуванням представлених вище методик кодування, а саме методики кодування/декодування з пріоритетами опишемо розроблений алгоритм перетворення алгоритму рою частинок.

Будемо вважати, що розмірність функції, що оптимізується – це кількість вузлів мережі, що досліджується. Тобто це означає, що кожна координата частинки відповідає певному вузлу заданої телекомунікаційної мережі.

В процесі побудови шляху з вузла телекомунікаційної мережі s до вузла мережі d перехід до наступного вузла з поточного розташування визначаємо, враховуючи структуру мережі, через ймовірність переходу в кожен вузол, що обчислюється як

$$p_i = 1 - \frac{x_i}{\sum_{k \in \{\text{сусіди вершини}\}} x_k},$$

де x_i – координата для даної вершини, $\{\text{сусіди вершини}\}$ – не відвідані сусідні вузли.

Таким чином в будь-який момент для будь-якої частинки ми можемо побудувати шлях в телекомунікаційній мережі, що вона собою представляє з застосуванням даної методики. Тобто даний підхід дозволяє застосовувати як стандартний, так і модифікований алгоритми рою частинок, які розроблені для оптимізації функції багатьох змінних для як задачі пошуку

найкоротшого шляху на графі, так і для задачі пошуку найкоротшого шляху в телекомунікаційній мережі [21].

Згідно з описаними методиками ми з легкістю в будь-який момент можемо декодувати отриманий шлях в телекомунікаційній мережі, проте нас цікавить в основному лише його довжина.

Підсумовуючи все описане вище можна представити фінальний варіант алгоритму рою частинок для задачі пошуку найкоротшого шляху в телекомунікаційній мережі:

Згенерувати початкову конфігурацію (для кожної частинки):
Виконати довільну ініціалізацію X_i
Виконати довільну ініціалізацію V_i
Обчислити $path_i$ за описаною методикою
Обчислити значення функції пристосованості $f(X_i)$
    $P_i \leftarrow \infty$
    $G \leftarrow \infty$
$iteration\_count \leftarrow 0$
// max_$iteration$ – максимальна кількість ітерацій

while ($iteration\_count$ < max_$iteration$)
    для кожної частинки i
        Обчислити V_i із застосуванням методів інерції швидкості та звуження множника
        Оновити X_i
        Обчислити $path_i$ за описаною методикою
        Обчислити значення функції пристосування $f(path_i)$
        if $f(path_i) < f(P_i)$
            $P_i \leftarrow path_i$

```
            if f(P_i) < f(G)
                G ← P_i
            Застосувати метод скидання швидкості
            Застосувати метод селективної регенерації


    iteration_count ← iteration_count + 1
end while

return G
```

Рис 2.2.8 – Псевдокод модифікованого алгоритму рою частинок з застосуванням описаної методики перетворення алгоритму для задачі пошуку найкоротших шляхів

3. ОПИС ПРОГРАМНОГО ДОДАТКУ

3.1. Середовище та компоненти розробки

Microsoft .NET – це технологія програмного забезпечення, розроблена корпорацією Microsoft у ролі платформи для розробки як консольних програм, так і веб-орієнтованих проектів. Дана технологія багато чим є продовженням ідей та принципів конкуруючої технології Java. Основною з ідей Microsoft. NET є сумісність її служб, що описані різними мовами програмування.

Розробка даної технології у 1999 році корпорацією Microsoft. Датою офіційного оголошення про розробку нової технології було 13.01.2000 р. Цього дня керівництво компанії озвучило нову стратегію, що мала назву Next Generation Windows Services. Всі існуючі розробки та розробки в майбутньому мали бути об’єднані цією стратегією у надання можливості користувачам працювати з мережею Інтернет, в тому числі за допомогою безпровідних пристроїв, що мають доступ до мережі, так само як це відбувається зі стаціонарних машин [17].

Програма, що розроблена з використанням .NET Framework, може бути написана будь-якою із доступних в .NET мов програмування, перш за все транслюється в проміжний код низького рівня (англ. Common Intermediate Language). Після цього проміжний код або виконується на віртуальній машині – міжмовним середовищем виконання (англ. Common Language Runtime – CLR), або ж транслюється утилітою NGen.exe в машинний код для конкретного цільового процесора. Використанню віртуальної машини надається перевага, оскільки це абстрагує розробників від вникання в апаратну частину обчислювальної машини. Якщо все ж надано перевагу використанню віртуальної машини CLR, то її JIT-компілятор в реальному часі (англ. just in time) виконає перетворення

проміжного коду в машинні коди цільового процесора. Сучасні технології динамічної компіляції дають змогу розробникам досягти високого рівня швидкодії. Віртуальна машина CLR також містить у собі керовані (англ. managed) компоненти керування безпекою, пам’яттю та обробки помилок (англ. exceptions), позбавляючи розробника турботи за ці речі.

Архітектуру .NET Framework можна знайти в міжмовній специфікації інфраструктури (англ. Common Language Infrastructure – CLI), що розроблена корпорацією Microsoft і затверджена ISO і ECMA. В даній специфікації описані типи даних платформи .NET, формат метаданих програмних модулів, систему правил виконання проміжного коду та інше.

Класи об’єктів .NET, є доступними усім мовам програмування CLR, і розміщені в бібліотеці класів фреймворку (англ. Framework Class Library – FCL). До даної бібліотеки належать класи технологій Windows Forms, ADO.NET, ASP.NET, LINQ, WPF, WCF та ін. Ядро бібліотеки називається базовою бібліотекою класів (англ. Base Class Library – BCL) [17].

Стек технологій платформи .NET представлений на рис. 3.1.1. [18]

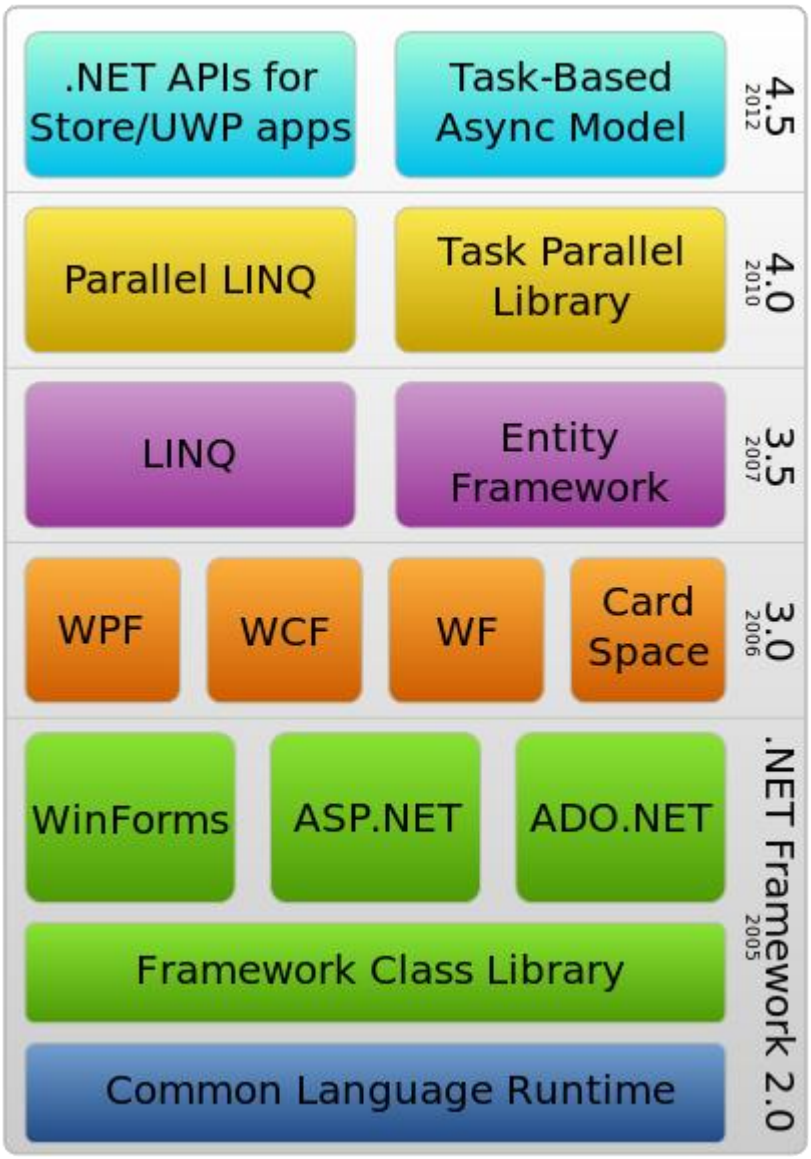


Рис. 3.1.1 – Стек технологій .NET Framework

Мова C# являє собою мову програмування з парадигмою строгої типізації, імперативною парадигмою, парадигмою декларативності, парадигмою функціональності, об'єктно-орієнтованою парадигмою (оскільки мова побудована на основі класів), а також з парадигмою компонент-орієнтованого програмування. Мова C# розроблена корпорацією Microsoft в рамках описаних вище змін щодо .NET, після чого була затверджена як стандарт ECMA (ECMA-334) і ISO (ISO / IEC 23270: 2006).

C# є однією з мов програмування, що належить до міжмовної специфікації інфраструктури.

Мова C# за своєю суттю є об’єктно-орієнтованою та універсальною мовою програмування. Команду, яка займається розробкою мови очолює Андреас Гейлсберг. На сьогоднішній день останньою версією мови є C# 7.2, яка була реалізована 13.11.2017 р.

Згідно стандарту ECMA, мета мови C# полягає в наступному:

- Призначення мови C# складаються з таких пунктів: мова має бути універсальною, сучасною, простою та об’єктно-орієнтованою мовою програмування.
- Мова та реалізації мови мають забезпечувати підтримку інженерних принципів та принципів побудови програмного забезпечення, таких як строга типізація, контроль границь індексів, виявлення спроб використання неініціалізованих змінних, а також автоматичного прибирання сміття (англ. garbage collection). Важливою складовою мови також є вбудоване забезпечення надійності, довговічності, і продуктивності праці розробника.
- Мова може бути використана для розробки програмних компонентів для розподілених середовищ.
- Портативність вихідного коду програми і зручність для пов’язаних з цим програмістів, особливо тих, які вже мають досвід роботи з мовами C або C++.
- Підтримка локалізації та різних культур.
- Мова C# є придатною для розробки програмного забезпечення як для систем, що виконуються на серверах (англ. hosted systems), так і вбудованих систем (англ. embedded systems), починаючи від значних за обсягом, що використовують операційні системи складного типу, і закінчуючи найменшими, що містять лише кілька рядків коду.

- Хоча програми C# і мають призначення бути економними в сенсі вимог до пам’яті та обчислювальної потужності комп’ютера, мова не може конкурувати за цими показниками з такими мовами як C чи асемблер.

Синтаксис мови є виразним та простим для вивчення. Розробники, які мали попередній досвід програмування з використанням мов C, C++ або Java з легкістю освоять синтаксис мови C#. Синтаксис мови C# робить простішим ті конструкції, які було складним в C++ і забезпечує багато можливостей, таких як nullable типи значень, перечислення (англ. enums), делегати, lambda-вирази, а також прямий доступ до пам’яті, яких немає в мові Java. Також мова C# підтримує універсальні методи і типи (англ. generics), що забезпечує більш високий рівень безпеки та продуктивності. Також мова C# підтримує ітератори (iterators), які дозволяють при реалізації колекцій об’єктів задавати власну поведінку при їх ітеруванні, що допомагає з легкістю використовувати їх в клієнтському коді в майбутньому. Вирази LINQ створенні для перетворення строго типізованих запиті на зручні конструкції мови.

Мова C# є об’єктно-орієнтованою мовою, тому містить реалізацію понять інкапсуляції, наслідування та поліморфізму. У визначення класів інкапсулюються всі змінні та методи програми, включно з методом Main – точкою входу в програму. Будь-який клас може безпосередньо наслідувати лише один батьківський клас, проте може також реалізовувати будь-яку кількість інтерфейсів. Методи, які виконують перевизначення віртуальних методів батьківського класу, мають йти з ключовим словом override.

Для мови C # структури (англ. struct) схожа на спрощену версію класів і являють собою тип, екземпляри якого, не враховуючи кілька винятків, розміщуються в стеці (англ. stack-allocated type). Структури при цьому

можуть реалізовувати інтерфейси, але не підтримують функціонал наслідування.

Не зважаючи на описані основні принципи об'єктно-орієнтованого програмування, розробка компонентів програмного забезпечення значно спрощується мовою C# завдяки кільком інноваційним конструкціям, які складаються з:

- делегатів – інкапсульованих сигнатур методів, які підтримують типізовано безпечні (англ. type-safe) повідомлення щодо поді1;
- властивості (англ. Properties), що виступають обгорткою методів доступу до прихованих полів;
- атрибути з декларативними метаданими щодо типу, які визначаються на етапі виконання часу виконання;
- XML-документація у вигляді вбудованих коментарів;
- Інтегровані запити мови (англ. LINQ), які забезпечують можливості запитів до різних джерел даних.

У випадку, коли програмісту необхідно забезпечити взаємодію з іншим програмним забезпеченням операційної системи Windows, таким як COM-об'єкти або власні бібліотеки Win32, мова C# підтримує процес під назвою Interop. Даний процес дозволяє програмам C# фактично виконувати будь-які команди, які могла б виконати цільова програма, описана мовою C++. Так само мова C# підтримує вказівники і таке поняття як небезпечний код спеціально для тих випадків, коли прямого доступу до пам'яті не можна уникнути.

Для програми мовою C# процес її побудови, якщо порівнювати з C і C++ є більш простим і гнучким. Немає необхідності створювати окремі файли заголовків (англ. headers), а типи і методи можуть бути оголошені в довільному порядку. У вихідному файлі програми мовою C# можна визначити будь-яку кількість структур, подій, інтерфейсів і класів.

Програма мовою C# виконується в описаному вище середовищі Microsoft .NET Framework.

Вихідний код, розроблений з використанням мови C#, компілюється в проміжну мову відповідно до згаданої вище специфікації CLI. Код проміжної мови, а також такі ресурси як текстові рядки (англ. strings) чи растрові зображення зберігаються на диску з розширенням EXE або DLL у більшості випадків в виконуваному файлі, що має назву збірки (англ. assembly). Кожна збірка також містить з маніфест, в якому знаходяться відомості про тип збірки, її версія, мова та регіональні параметри, а також вимоги до безпеки.

При виконанні програмного забезпечення на мові C# збірка завантажується в виконуване середовище відповідно до відомостей у маніфесті. Після цього, якщо дотримано вимоги безпеки, середовище CLR виконує компіляцію «на льоту» для перетворення проміжного коду в команди машинного коду. Виконуване середовище CLR також виконує керування іншими службами, до яких відносяться обробка виключень (англ. exceptions), автоматичний збір сміття та керування ресурсами. Код, що виконується середовищем CLR також має назву керованого коду на відміну від некерованого коду, який може компілюватися лише в машинний код, призначений для певної цільової системи. Рисунок 3.1.2 містить графічне представлення залежності між файлами з вихідним кодом мовою C#, бібліотеками класів .NET Framework, збірками і виконуваним середовищем на час компіляції і час виконання.

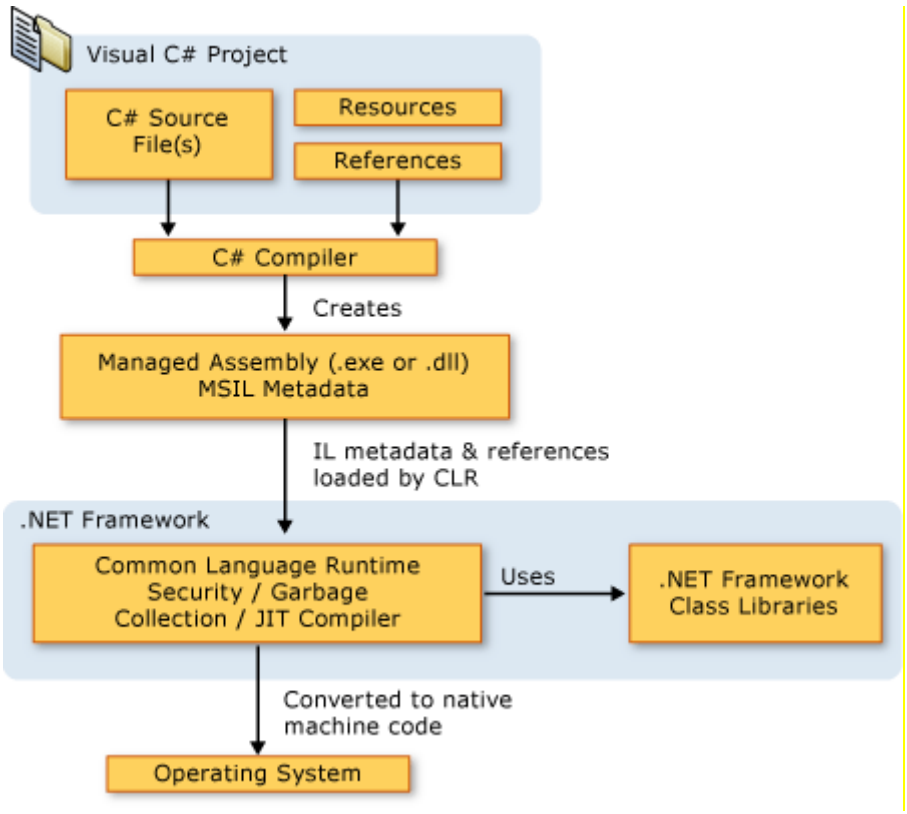


Рис. 3.1.2 – Залежності компонентів при компіляції програми мовою C# [17]

Однією з ключових особливостей технології .NET Framework є сумісність мови C#. Оскільки код проміжний код створюється за допомогою компілятора C# і відповідає специфікації загальних типів (англ. Common Type Specification) то проміжний, який є згенерованим для програми мовою C#, може виконуватись поряд з кодом, який був написаний на мові Visual Basic, Visual C++, або будь-якою з більш ніж 20 сумісних згідно специфікації мов. Один компонент може містити у собі кілька модулів, написаних різними мовами .NET Framework, а описані типи можуть посилатися один на одного таким самим чином, неначе вони розроблені одною мовою.

На додаток до сервісу часу виконання, технологія .NET Framework також містить бібліотеку на даний момент з більш ніж 4000 класів, організованих у відповідних просторах імен (англ. namespaces), які надають

розробнику широкий діапазон корисних функцій для більшості життєвих випадків, починаючи введенням і виведенням файлів й завершуючи маніпуляціями з рядками у форматі XML та керуванням формами програми технології минулого Windows Forms.

Microsoft Visual Studio – серія продуктів корпорації Microsoft, яка містить у собі інтегроване середовище розробки програмного забезпечення, а також інші інструменти розробки. Ця серія дозволяє виконувати розробку як консольних програм, так і програм з графічним інтерфейсом, в тому числі з підтримкою застарівших технологій, таких як Windows Forms, а також web-сайти, web-служби як базовою мовою C#, так і з використанням інших керованих кодів для всіх платформ, підтримуваних Microsoft, а це: Windows Mobile, Windows CE, .NET Framework, Xbox, Windows Phone .NET Compact Framework та Silverlight.

Microsoft Visual Studio містить у собі редактор вихідного коду з підтримкою технології IntelliSense (підказки у реальному часі), а також можливістю серйозного рефакторингу (англ. refactoring) коду. Вбудовані компоненти для відладки (англ. debugging) надають змогу працювати як у ролі відладки рівня вихідного коду, так і у ролі відладки рівня операційної системи. Всі інші вбудовані інструменти складаються з редактора форм для спрощення розробки графічного інтерфейсу програми, веб-редактор, дизайнер структури класів і схем бази даних. Microsoft Visual Studio дозволяє виконувати розробку та підключення сторонніх програм (плагінів) для доповнення функціональності на всіх рівнях, включаючи підтримку систем контролю версій вихідного коду таких як Subversion і Visual SourceSafe, Team Foundation Server, Git, а також додавати нові набори інструментів (прикладом може бути редагування і візуальне проектування коду на предметно-орієнтованих мовах програмування).

В процесі розробки програмного забезпечення для перевірки наукової гіпотези, з метою збереження історичності коду та запобігання можливих його втрат використовувалася система контролю версій GitHub.

GitHub є одним з найбільших web-сервісів для виконання командної розробки програмного забезпечення. Існує безкоштовний та платні плани користування сервісом. GitHub в своїй основі містить систему керування версіями Git і розроблений з використанням технології Ruby on Rails і Erlang корпорацією GitHub Inc.

Сервіс є безкоштовним для проектів з відкритим вихідним кодом та надає своїм користувачам усі доступні можливості для підтримки відкритості.

Репозиторій програмних засобів, розроблених для магістерської дисертації знаходиться за посиланням

https://github.com/MaxBondarchuk/PSO-Open-Shortest-Path.

Оскільки в програмі магістерської дисертації наявні компоненти з реалізацією через універсальні типи (англ. generics), то надати інформацію про дані типи.

Універсальні шаблони є частиною мови C# починаючи з версії 2.0. Універсальні шаблони представляють концепцію параметрів типів в технології .NET Framework та дозволяють виконувати розробку методів та класів, що не дотримуються прив’язки до визначених типів до того часу, поки метод або клас не буде використано в клієнтському коді і не буде створено його екземпляр. Для прикладу, з використанням параметру універсального типу T можна створити окремий клас, який код клієнтської програми зможе використовувати без ризику виникнення помилок під час виконання або операцій приведення типів.

Основними можливостями універсальних шаблонів є:

- Досягнення максимального рівня повторного використання коду, безпека типів та продуктивність через зменшення інформації про типи в оперативній пам’яті.
- Найпоширенішим випадком застосування універсальних шаблонів є використання колекцій класів.
- Бібліотека класів FCL платформи .NET Framework містить певну кількість універсальних класів колекцій в просторі імен System.Collections.Generic. Вони можуть використовуватися замість таких класів як ArrayList в просторі імен System.Collections й надавати переваг у продуктивності програм.
- Розробник може створювати власні універсальні класи, інтерфейси, події, методи і делегати.
- До типів даних можуть застосовуватися обмеження доступу до методів.
- З використанням рефлексії є можливість отримати дані щодо типів, які використовуються в універсальному типі даних.

Оскільки в даній науковій роботі програма для перевірки нового алгоритму рою частинок для задачі пошуку найкоротших шляхів в телекомунікаційних мережах використовується останнє покоління платформи .NET: .NET Core, то варто сказати кілька слів про дану технологію.

Технологія **.NET Core** є безкоштовним багатоплатформним (англ. cross-platform) фреймворком, що використовує керований код (англ. managed code) та підтримується такими операційними системами як Windows, Linux та Mac OSX. Основною відмінністю від .NET Framework є те, що код (англ. source code) .NET Core є повністю відкритим і доступний

за наступним посиланням https://github.com/dotnet/core. .NET Core містить CoreCLR, що є за своєю суттю крос-платформною реалізацією CLR, тобот віртуальною машиною, що керує виконанням програм в середовищі .NET. CoreCLR містить оптимізований «компілятор на льоту» (англ. just-in-time compiler) під назвою RyuJIT. .NET Core також включає в себе CoreFX, що є частковим відгалуженням FCL (англ. Framework Class Library – стандартна бібліотека класів платформи .NET). Відповідно, реалізації всіх класів Core також знаходяться у вільному доступі за посиланням https://github.com/dotnet/corefx. На той момент як .NET Core є підмножиною API .NET Framework, він містить також власний API (англ. application programming interface – прикладний програмний інтерфейс), який не входить до складових .NET Framework. Крім того .NET Core містить компонент CoreRT, що оптимізований під інтеграцію з бінарними файлами AOT-компіляції (англ. ahead-of-time compilation – компіляція перед виконанням). UWP (англ. Universal Windows Platform – універсальна платформа Windows) також використовує один з варіантів бібліотеки .NET Core. UWP – це платформа, яка створена корпорацією Microsoft і була вперше представлена в операційній системі Windows 10. Метою даної платформи є представлення бази для розробки універсальних додатків Windows, що запускаються як на операційних системах покоління Windows 10, так і на операційних системах Windows 10 Mobile без жодних змін для програмного коду. .NET Core містить інтерфейс командного рядка (англ. command line interface), який пропонує місце входу для операційних систем і надає можливості компіляції і пакетів керування для розробників.

.NET Core містить 4 крос-платформні сценарії: веб-додатки ASP.NET Core, консольні додатки, бібліотеки та UWP-програми. .NET Core підтримує технології Windows Forms або WPF, які відповідають за створення стандартного графічного інтерфейсу для персональних комп’ютерів на

операційній системі Windows. .NET Core є також модульним фреймворком, що означає, що замість бібліотек (англ. assembly), розробники мають справу з NuGet-пакетами. На відміну від .NET Framework, який оновлюється службою Windows Update, .NET Core оновлюється менеджером пакетів при завантаженні оновлень [19].

3.2. Опис основної програми

З метою перевірки наукової гіпотези на практиці розроблено програмні засоби, що дозволяють порівнювати результати роботи базового та модифікованого алгоритмів рою частинок для задачі пошуку найкоротших шляхів у телекомунікаційних мережах.

Зазначена програма працює лише в консольному режимі, оскільки це дозволяє виконувати більше математичних обчислень за одиницю часу роботи процесора і не втрачати час на асинхронні операції візуального відображення результатів поточної оптимізації в режимі реального часу.

Процес виконання програми (її «головне вікно») бачимо на рисунку 3.2.1.

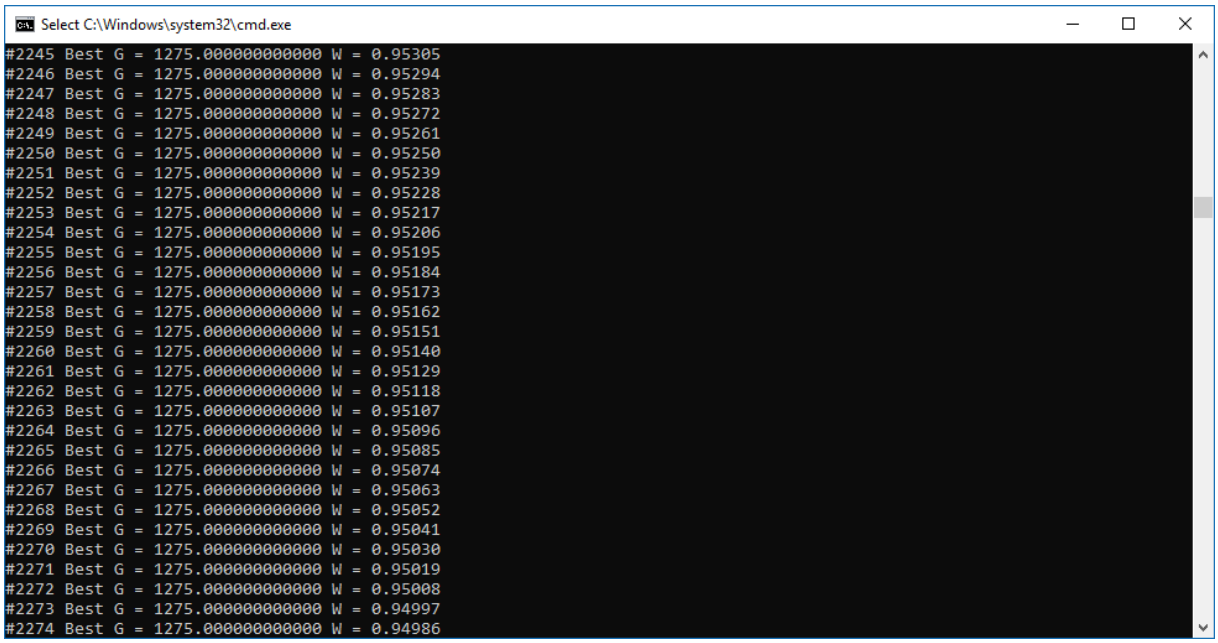


Рис. 3.2.1 – Головне вікно програми

Для коректної роботи програми важливо задати початкову конфігурацію телекомунікаційної мережі, для якої відбуватиметься пошук найкоротшого шляху з її першого до останнього вузла.

Файл, що містить конфігурацію телекомунікаційної мережі має мати формат *.pso, що є скороченою назвою алгоритму рою частинок.

Даний файл повинен мати наступну структуру свого змісту:

- c <коментар> – вказує, що даний рядок файлу є коментарем і зазвичай використовується для задання заголовків файлів, що містять додаткову інформацію для науковця;
- p <число> – вказує, що даний рядок містить інформацію щодо кількісті вузлів телекомунікаційної мережі, що задана у даному файлі;
- i <масив розмірності p> – вказує, що даний рядок містить інформацію про вузол телекомунікаційної мережі. В цьому випадку за латинською буквою «i» має бути розміщений масив цілих чисел, що відповідають за вартості каналів зв’язку (ребер) між даним вузлом та всіма іншими вузлами мережі в порядку зростання їх індексів (у порядку, в якому вони задані у файлі). У випадку, якщо між вузлами немає каналу зв’язку (що є звичним явищем для телекомунікаційної мережі), то необхідно поставити нуль. Важливо, що в даному рядку має знаходитись саме така кількість чисел, яка дорівнює кількості вузлів мережі.

Важливо зазначити деякі правила заповнення даного файлу з конфігурацією телекомунікаційної мережі.

Перш за все, рядки з коментарями можуть знаходитись у будь-якому місці файлу, оскільки дані рядки ігноруються парсером.

По-друге, перед першим рядком, що починається латинською буквою «i» має знаходитись рядок, що починається на букву «p», який містить інформацію щодо кількісті вузлів мережі. Якщо цю умову не буде виконано, парсер не зможе бути в майбутньому впевненим, скільки чисел необхідно зчитувати для інформації щодо каналів зв’язку.

Парсер файлів *.pso (програмний модуль, що виконує зчитування текстовою інформації з файлу та виконує перетворення її у структуру екземплярів класів в оперативній пам’яті обчислювальної машини, що відповідають мережі) працює за допомогою регулярних виразів.

Регулярний вираз (англ. regular expression, скорочено regex або regexp) – це рядок, що описує або збігається з множиною рядків, відповідно до набору спеціальних синтаксичних правил. Вони використовуються в багатьох текстових редакторах та допоміжних інструментах для пошуку та зміни тексту на основі заданих шаблонів. Мова програмування C# (як і більшість інших сучасних мов програмування), якою розроблені програмні засоби для перевірки наукового дослідження, має вбудовані в .NET Framework засоби для роботи з регулярними виразами.

З метою кращого розуміння структури файлу з конфігурацією мережі наведемо приклад такого файлу.

```
c FILE: psoConfig10.pso
c
c SOURCE: Maksym Bondarchuk (bondarchuk.m.y@gmail.com)
c
c DESCRIPTION: This is an example file for diploma project
c first line after comment (p) gives problem's size
c
p 10
i 0 5 0 362 325 0 249 0 282 0
i 5 0 443 2 291 26 6 360 58 119
i 0 443 0 68 0 1 17 164 0 366
i 362 2 68 0 0 3 0 235 0 437
```

```
i 325 291 0 0 0 187 0 11 0 169
i 0 26 185 1 187 0 337 375 322 399
i 249 6 17 0 0 337 0 52 196 36
i 0 360 164 235 11 375 52 0 427 112
i 282 58 0 0 0 322 196 427 0 279
i 0 119 366 437 169 399 36 112 279 0
```

Рис 3.2.2 – Файл psoConfig10.pso

В загальному, програма складається з кількох логічних частин.

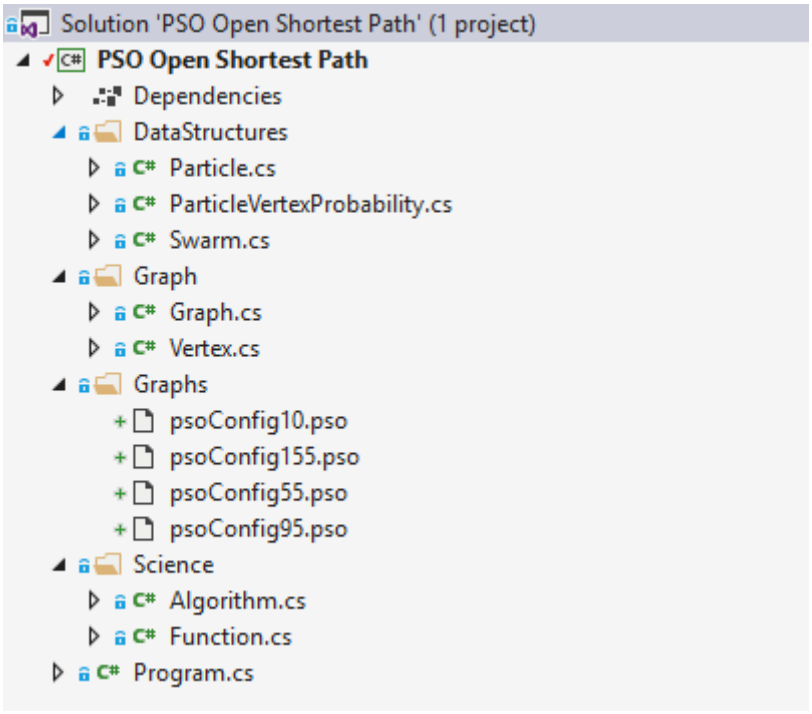


Рис 3.2.3 – Загальна структура програми

Перша частина класів (Graph), що описують власне структуру телекомунікаційної мережі, до якою належать наступні класи:

- Graph.cs;
- Vertex.cs.

Подібною до першої частини є друга частина (DataStructures), що описує структури даних для власне алгоритму рою частинок. До цієї частини належать такі класи:

- Particle.cs;
- ParticleVertexProbability.cs;
- Swarm.cs.

Третя частина програми описує математичну логіку алгоритму рою частинок, а саме: ініціалізацію згідно викладеної вище теорії та всі кроки основної частини. Дана частина містить наступний набір класів:

- Algorithm.cs;
- Function.cs.

Згідно до викладеної вище теоретичної частини щодо технології .NET Framework, кожна консольна програма повинна мати точку програми, то для нас цим місцем є клас Program.cs.

Опишемо класи програми більш детально.

Клас Graph.cs

Даний клас містить кістяк структури телекомунікаційної мережі, тобто множину вузлів даної телекомунікаційної мережі, а також метод для завантаження структури телекомунікаційної мережі з файлу в оперативну пам’ять обчислювальної машини з використанням регулярних виразів згідно процедури, що описана вище.

Фрагмент коду, що представляє даний клас, зображено на рисунку 3.2.4.

```
6 references | Maksym Bondarchuk, 174 days ago | 1 author, 3 changes
public class Graph
{
    public readonly List<Vertex> Vertices = new List<Vertex>();

    1 reference | Maksym Bondarchuk, 174 days ago | 1 author, 1 change
    public bool LoadFromFile(string fileName)
    {
        using (var sr = new StreamReader(fileName))
        {
            string line;
            var problemSize = -1;
            while ((line = sr.ReadLine()) != null)
            {
                if (line.Length == 0) { continue; }

                if (line[0] == 'p')
                    problemSize = Convert.ToInt32(Regex.Match(line, @"\d+").Value);

                if (line[0] == 'i')
                {
                    MatchCollection matches = Regex.Matches(line, @"\d+");

                    if (matches.Count != problemSize) { return false; }

                    var vertex = new Vertex();
                    for (var i = 0; i < matches.Count; i++)
                    {
                        var weight = Convert.ToInt32(matches[i].Value);
                        vertex.Weights.Add(weight);
                    }
                    Vertices.Add(vertex);
                }
            }
        }

        return true;
    }
}
```

Рис 3.2.4 – Фрагмент коду класу Graph.cs

Клас Vertex.cs

Даний клас містить необхідні компоненти для завершення побудови структури телекомунікаційної мережі та є логічним доповненням до класу Graph.cs.

Згідно з описаною вище процедурою подання мережі в файлі на диску, даний клас містить множину вартостей (ваг) всіх можливих каналів зв'язку між вузлами мережі.

Фрагмент коду, що представляє даний клас, зображено на рисунку 3.2.5.

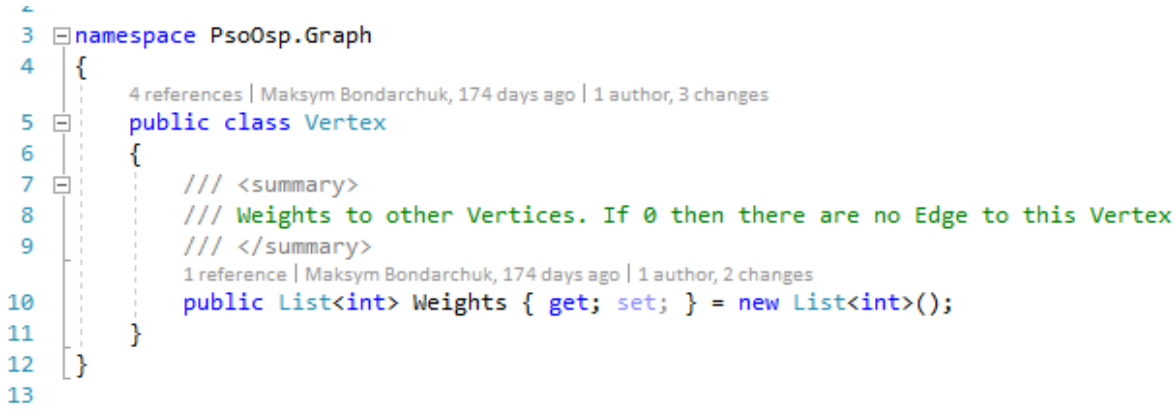


Рис 3.2.5 – Фрагмент коду класу Vertex.cs

Клас Particle.cs

Даний клас містить необхідні компоненти для опису однієї частинки з рою частинок, що є основою біологічної моделі алгоритму. Частинка містить такі обов'язкові компоненти:

- Набір дійсних чисел X – пріоритети вузлів телекомунікаційної мережі, згідно до методики перетворення, описаної в частині 2 наукової роботи.
- Дійсне число Fx – значення функції пристосованості частинки, що описується даним класом.
- Набір дійсних чисел V – швидкості частинки по кожній її координаті (з урахуванням методики перетворення алгоритму – набору пріоритетів вузлів).

- Набір дійсних чисел P – точна копія множини X на момент, коли значення функції пристосованості даної частинки було найкращим за всю історію її існування.
- Дійсне число Fp – найкраще значення функції пристосованості частинки за всю історію її існування з моменту ініціалізації. Слід зауважити, що при застосуванні методу скидання швидкості, дане поле також оновлюється математичним значенням ∞ (оскільки комп'ютер працює з числами в скінченному представленні, то використовується значення мови C# – int.MaxValue).
- Метод $UpdateP$, що виконує фіксування в пам'яті поточного розташування частинки в просторі пошуку як потенційний розв'язок задачі пошуку найкоротшого шляху.
- Метод $Generate$, що виконує генерацію частинки довільним чином в заданій телекомунікаційній мережі на етапі ініціалізації.

Фрагмент коду, що представляє даний клас, зображено на рисунку 3.2.6.

```
5 references | Maksym Bondarchuk, 174 days ago | 1 author, 3 changes
public class Particle
{
    /// <summary>
    /// Coordinates of particle in search-space
    /// </summary>
    13 references | Maksym Bondarchuk, 174 days ago | 1 author, 2 changes
    public List<double> X { get; } = new List<double>();

    /// <summary>
    /// Function value for X set (for optimization)
    /// </summary>
    5 references | Maksym Bondarchuk, 174 days ago | 1 author, 3 changes
    public int Fx { get; set; } = int.MaxValue;

    /// <summary>
    /// Velocity
    /// </summary>
    9 references | Maksym Bondarchuk, 174 days ago | 1 author, 2 changes
    public List<double> V { get; } = new List<double>();

    /// <summary>
    /// Particle best location
    /// </summary>
    7 references | Maksym Bondarchuk, 174 days ago | 1 author, 2 changes
    public List<double> P { get; } = new List<double>();

    /// <summary>
    /// Particle best value
    /// </summary>
    5 references | Maksym Bondarchuk, 174 days ago | 1 author, 3 changes
    public int Fp { get; private set; } = int.MaxValue;

    1 reference | Maksym Bondarchuk, 174 days ago | 1 author, 2 changes
    public void UpdateP()
    {
        P.Clear();
        P.AddRange(X);
        Fp = Fx;
    }

    2 references | Maksym Bondarchuk, 174 days ago | 1 author, 1 change
    public void Generate(Graph.Graph graph, Function func, Random random)
    {
        X.Clear();
```

Рис 3.2.6 – Фрагмент коду класу Vertex.cs

Клас ParticleVertexProbability.cs

Даний клас містить є класом-розв’язкою між класами Particle та Vertex, що являє собою реалізацію відношення «багато-до-багатьох» та

містить інформацію щодо ймовірностей (пріоритетів) вузлів телекомунікаційної мережі згідно алгоритму.

Даний клас містить наступні компоненти:

- Посилання на вузол телекомунікаційної мережі $Vertex$.
- Дійсне число $Probability$, що є пріоритетом для даної вершини.

Фрагмент коду, що представляє даний клас, зображено на рисунку 3.2.7.

```
namespace PsoOsp.DataStructures
{
    0 references | Maksym Bondarchuk, 174 days ago | 1 author, 1 change
    public class ParticleVertexProbability
    {
        0 references | Maksym Bondarchuk, 174 days ago | 1 author, 1 change
        public Vertex Vertex { get; set; }

        0 references | Maksym Bondarchuk, 174 days ago | 1 author, 1 change
        public double Probability { get; set; }
    }
}
```

Рис 3.2.7 – Фрагмент коду класу ParticleVertexProbability.cs

Клас Swarm.cs

Даний клас містить у собі біологічну модель рою частинок, тобто посилання на всі частинки з рою, а також допоміжні компоненти для роботи алгоритму:

- Набір посилань на частинки рою $Particles$ – частинки відповідно до розділу 2.
- Набір дійсних чисел G – точна копія множини X на момент, коли значення функції пристосованості найкращої частинки на певний момент часу (ітерації) було найкращим серед всіх частинок за всю історію її існування.
- Дійсне число Fg – найкраще значення функції пристосованості всіх частинок за всю історію їх існування з моменту ініціалізації.

- Метод $UpdateG$, що виконує фіксування в пам’яті з поточного розташування певної частинки рою в просторі пошуку, що передається вхідним параметром як потенційний розв’язок задачі пошуку найкоротшого шляху.

Фрагмент коду, що представляє даний клас, зображено на рисунку 3.2.8.

```
2 references | Maksym Bondarchuk, 174 days ago | 1 author, 2 changes
public class Swarm
{
    3 references | Maksym Bondarchuk, 174 days ago | 1 author, 2 changes
    public List<Particle> Particles { get; } = new List<Particle>();

    /// <summary>
    /// Swarm's best known position
    /// </summary>
    4 references | Maksym Bondarchuk, 174 days ago | 1 author, 2 changes
    public List<double> G { get; } = new List<double>();

    /// <summary>
    /// Function value for G set (for optimization)
    /// </summary>
    4 references | Maksym Bondarchuk, 174 days ago | 1 author, 2 changes
    public double Fg { get; private set; } = double.MaxValue;


    2 references | Maksym Bondarchuk, 174 days ago | 1 author, 1 change
    public void UpdateG(IEnumerable<double> p, Function f, Graph.Graph graph)
    {
        G.Clear();
        G.AddRange(p);
        Fg = f.F(G, graph);
    }
}
```

Рис 3.2.8 – Фрагмент коду класу Swarm.cs

Клас Algorithm.cs

Даний клас містить у собі власне реалізацію алгоритму рою частинок: біологічну структуру рою частинок, математичну структуру телекомунікаційної мережі (представлення у вигляді графа), а також повну модель соціальної поведінки частинок.

Даний клас складається з наступних компонентів:

- Дійсної константи $PhiP$, що є одним з параметрів, що задає модель когнітивної поведінки алгоритму з нахилом до власного незалежного розвитку частинки як це було описано вище.
- Дійсної константи $PhiG$, що є одним з параметрів, що задає модель когнітивної поведінки алгоритму з нахилом до розвитку частинки під впливом інших частинок як це було описано вище.
- Об’єкт класу $Function\ Func$, що містить у собі посилання на функцію для оптимізації алгоритмом рою частинок. Дана функція має містити процес перетворення задачі оптимізації на задачу оптимізації в сенсі задачі пошуку найкоротших шляхів.
- Об’єкт класу $Random\ Random$, що є основою для генерування псевдодовільних чисел для розроблюваного алгоритму рою частинок (для основних формул, а також для ініціалізацій). Зауважимо, що у випадку, якщо в назві об’єкту кілька разів повторюється одне слово, то це означає, що ідентифікатор поля збігається з назвою класу, що допустимо в мові C#, а також назва може збігатися з простором імен (англ. namespace).
- Об’єкт $Graph.Graph\ Graph$, що містить структуру досліджуваної телекомунікаційної мережі і описаний вище.
- Конструктор $Algorithm$ з параметрами, які необхідні для коректної роботи алгоритму, який виконує ініціалізацію ро частинок та інших компонентів алгоритму.
- Метод Run, що є кістяком та містить у собі основну частину алгоритму рою частинок. Метод працює з вже ініціалізованим роєм та готовою структурою мережі.

Фрагмент коду, що представляє даний клас, зображено на рисунку 3.2.9.

```
2 references | Maksym Bondarchuk, 174 days ago | 1 author, 2 changes
public class Algorithm
{
    #region Constants: private
    private const double PhiP = 1;
    private const double PhiG = 1;
    #endregion

    #region Properties: private
    9 references | Maksym Bondarchuk, 176 days ago | 1 author, 1 change
    private Swarm Swarm { get; } = new Swarm();
    16 references | Maksym Bondarchuk, 174 days ago | 1 author, 2 changes
    private Function Func { get; }

    5 references | Maksym Bondarchuk, 176 days ago | 1 author, 1 change
    private Random Random { get; } = new Random();

    6 references | Maksym Bondarchuk, 174 days ago | 1 author, 1 change
    private Graph.Graph Graph { get; }
    #endregion

    1 reference | Maksym Bondarchuk, 174 days ago | 1 author, 1 change
    public Algorithm(Graph.Graph graph, int swarmSize, Function func)
    {
        Func = func;
        Graph = graph;

        Swarm.Particles.Clear();
        for (var i = 0; i < swarmSize; i++)
        {
            var particle = new Particle();
            particle.Generate(Graph, Func, Random);
            Swarm.Particles.Add(particle);


            if (particle.Fp < Swarm.Fg)
            {
                Swarm.UpdateG(particle.P, Func, Graph);
            }
        }
    }
```

Рис 3.2.9 – Фрагмент коду класу Algorithm.cs

Клас Function.cs

Даний клас містить у собі реалізацію функції перетворення задачі оптимізації для задачі пошуку найкоротших шляхів.

Фрагмент коду, що представляє даний клас, зображено на рисунку 3.2.10.

```
9 references | Maksym Bondarchuk, 175 days ago | 1 author, 3 changes
public class Function
{
    /// <summary>
    /// Function
    /// </summary>
    8 references | Maksym Bondarchuk, 175 days ago | 1 author, 3 changes
    public Func<List<double>, Graph.Graph, int> F { get; set; }

    /// <summary>
    /// Lower boundary
    /// </summary>
    7 references | Maksym Bondarchuk, 177 days ago | 1 author, 1 change
    public double BoundLower { get; set; }

    /// <summary>
    /// Upper boundary
    /// </summary>
    7 references | Maksym Bondarchuk, 177 days ago | 1 author, 1 change
    public double BoundUpper { get; set; }

    6 references | Maksym Bondarchuk, 177 days ago | 1 author, 1 change
    public double Dimensions { get; set; }

    6 references | Maksym Bondarchuk, 177 days ago | 1 author, 1 change
    public double KillProbability { get; set; }



    1 reference | Maksym Bondarchuk, 175 days ago | 1 author, 1 change
    public double MinVelocity { get; set; } = -1.2;
    4 references | Maksym Bondarchuk, 177 days ago | 1 author, 1 change
    public double MaxVelocity { get; set; } = 1.2;

    3 references | Maksym Bondarchuk, 177 days ago | 1 author, 1 change
    public int IterationsNumber { get; set; } = 10000;
}
```

Рис 3.2.10 – Фрагмент коду класу Function.cs

Всі інші дії виконуються в класі Program.cs, що є точкою входу в програму мовою C# з використанням .NET Framework та .NET Core. Тобто з методу Main цього класу виконується запуск алгоритму, а перед цим відбувається зчитування та обробка конфігурації телекомунікаційної мережі з текстового файлу.

Також даний клас відповідає за діалог з користувачем, як це представлено на рисунку 3.2.11.

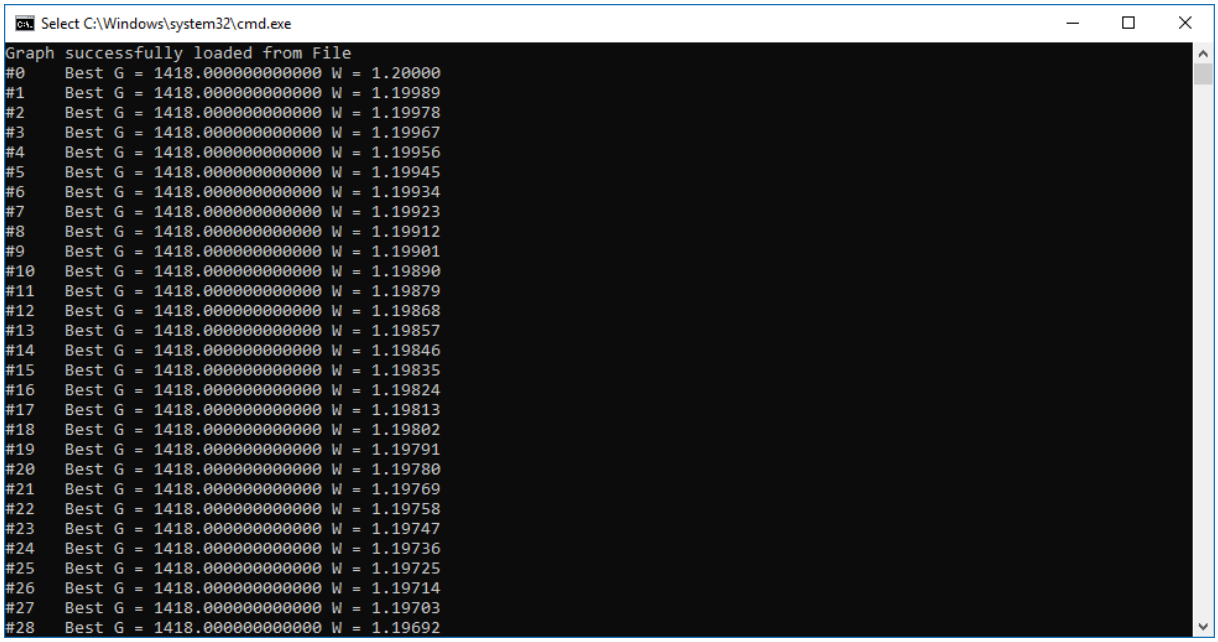


Рис 3.2.11 – Фрагмент діалогу з користувачем

4. РЕЗУЛЬТАТИ ТЕСТУВАННЯ ЗАПРОПОНОВАНОГО МЕТОДУ

4.1. Опис експериментів

Оскільки метою даної роботи є наукова та програмна розробка модифікацій алгоритму рою частинок та його застосування до задачі пошуку найкоротшого шляху в телекомунікаційній мережі, то тести будемо виконувати для оригінальної версії алгоритму, а також для кожної з модифікацій окремо, та для всіх модифікацій разом з метою з'ясування відсотку впливу кожної модифікації на алгоритм.

З метою отримання якомога найбільш об'єктивних даних, для кожного наведеного нижче результату запуск програми відбувався 20 разів, а наведене значення є усередненим.

Оскільки даний метод є метаевристичним та належить до такого типу алгоритмів, що не гарантують обов'язковість отримання найкращого з можливих значення, то вводиться поняття показника успішності (англ. success rate). Показник успішності розраховується за наступною формулою:

$$SR = \frac{success}{success + fail},$$

де $success$ – кількість успішних завершень алгоритму, $fail$ – кількість неуспішних завершень алгоритму.

Тобто вважається, що заздалегідь відомий результат роботи алгоритму (довжина найкоротшого шляху в телекомунікаційній мережі), про який алгоритм не знає і при завершенні його роботи отриманий результат порівнюється з необхідним. Для обчислення показника успішності в процесі тестування ми будемо використовувати 100 запусків алгоритму.

З описаного вище робимо висновок, що ідеальним (максимальним) значенням показника успішності є 1, чого ми й намагатимемося досягти, і

чим більшим буде значення показника успішності, тим якіснішим буде наш алгоритм.

Також варто зазначити, що всі версії методу були розроблені самостійно в програмі, що описана вище та не використовують запозичень коду, тому отримані результати можуть відрізнятись від результатів отриманих іншими дослідниками.

Для отримання даних щодо кожної з версій методу будемо виконувати виміри таких величин: показника успішності, кількості ітерацій та часу виконання алгоритму до його зупинки. Оскільки даний метод запрограмований на певну кількість ітерацій, то щоб оцінити реальну кількість ітерацій, необхідну методу для досягнення ним збіжності будемо вважати, що алгоритм отримав розв'язок у тому випадку, якщо за останні 5000 ітерацій не було отримано жодних змін, а максимально допустиму кількість ітерацій встановимо рівною 100000.

Для тестування методів будемо використовувати десять різних топологій телекомунікаційних мереж з різною кількістю вузлів. Для всіх мереж будемо використовувати кількість каналів зв'язку між вузлами, що обчислюється за формулою:

$$e = \frac{n^2}{5},$$

де n – кількість вузлів мережі.

Тобто кожен вузол мережі в середньому з'єднаний з 20% від загальної кількості вузлів мережі.

Оскільки модифікована селективна регенерація вимагає використання значення φ_p бути меншим за значення φ_g, то для всіх запусків методу будемо використовувати значення $\varphi_p = 0{,}5$ і $\varphi_g = 0{,}75$.

Для всіх запусків програми будемо використовувати популяцію кількістю 75 частинок.

З отриманих результатів необхідно зробити висновок про стабільність і стійкість модифікацій.

Таблиця 4.1.1 – Результати роботи оригінального методу рою частинок

Номер топології	**Кількість вузлів**	**Кількість ітерацій**	**Час, секунд**	**SR**
1	50	31807	1.0501	0.8655
2	60	38366	1.3212	0.8207
3	70	45313	1.5025	0.8281
4	80	51660	1.6585	0.8260
5	90	57996	1.9313	0.8298
6	100	63705	2.1043	0.8646
7	110	71444	2.3281	0.8571
8	120	77537	2.5821	0.8701
9	130	84450	2.8446	0.8750
10	140	89452	2.9458	0.8361

де SR – показник успішності алгоритму.

Таблиця 4.1.2 – Результати роботи методу рою частинок з використанням інерції швидкості

Номер топології	**Кількість вузлів**	**Кількість ітерацій**	**Час, секунд**	**SR**
1	50	31604	1.0545	0.8653
2	60	37656	1.2221	0.8550
3	70	43881	1.5087	0.8813
4	80	50597	1.6871	0.8612
5	90	57207	1.8616	0.8554

Номер топології	Кількість вузлів	Кількість ітерацій	Час, секунд	SR
6	100	63587	2.0419	0.8674
7	110	69950	2.3170	0.8602
8	120	76037	2.5115	0.8593
9	130	82622	2.7353	0.8659
10	140	87964	3.0272	0.8952

де SR – показник успішності алгоритму.

Таблиця 4.1.3 – Результати роботи методу рою частинок з використанням звуження множника

Номер топології	Кількість вузлів	Кількість ітерацій	Час, секунд	SR
1	50	31849	1.0181	0.8865
2	60	37816	1.2491	0.8492
3	70	43688	1.5105	0.8492
4	80	50221	1.6383	0.8785
5	90	56905	1.8939	0.8575
6	100	63077	2.0576	0.8565
7	110	69981	2.3240	0.8956
8	120	76349	2.5454	0.8889
9	130	82568	2.7600	0.8395
10	140	87655	2.8943	0.8441

де SR – показник успішності алгоритму.

Таблиця 4.1.4 – Результати роботи методу рою частинок з використанням скидання швидкості

Номер топології	**Кількість вузлів**	**Кількість ітерацій**	**Час, секунд**	**SR**
1	50	31558	1.0177	0.8578
2	60	37317	1.2396	0.8452
3	70	44169	1.5103	0.8731
4	80	50284	1.6984	0.8438
5	90	56297	1.8853	0.8720
6	100	63357	2.1052	0.8686
7	110	69015	2.3560	0.8965
8	120	75378	2.4476	0.8968
9	130	80904	2.6964	0.8946
10	140	87094	2.8381	0.8680

де SR – показник успішності алгоритму.

Таблиця 4.1.5 – Результати роботи методу рою частинок з використанням селективної регенерації

Номер топології	**Кількість вузлів**	**Кількість ітерацій**	**Час, секунд**	**SR**
1	50	30962	1.0104	0.8781
2	60	37420	1.2649	0.8587
3	70	43971	1.4124	0.8866
4	80	49857	1.6670	0.8498
5	90	56511	1.8543	0.8901
6	100	62052	2.0416	0.8957
7	110	68488	2.3005	0.8861

Номер топології	Кількість вузлів	Кількість ітерацій	Час, секунд	SR
8	120	74938	2.4679	0.8547
9	130	81913	2.7342	0.8870
10	140	87461	2.8446	0.8500

де SR – показник успішності алгоритму.

Таблиця 4.1.6 – Результати роботи методу рою частинок з використанням всіх запропонованих модифікацій

Номер топології	Кількість вузлів	Кількість ітерацій	Час, секунд	SR
1	50	29524	0.9761	0.9008
2	60	35790	1.1499	0.9164
3	70	41565	1.3619	0.9161
4	80	46972	1.5886	0.9269
5	90	53338	1.7668	0.9054
6	100	58546	1.9807	0.9390
7	110	64916	2.1133	0.9218
8	120	70521	2.4173	0.9018
9	130	77269	2.5703	0.9224
10	140	82942	2.8482	0.9099

де SR – показник успішності алгоритму.

З метою більш наочного аналізу даних, представимо дані з отриманих експериментів у вигляді графіків.

Схема 4.1.1 – Залежність усередненої кількості ітерацій методу від номеру топології

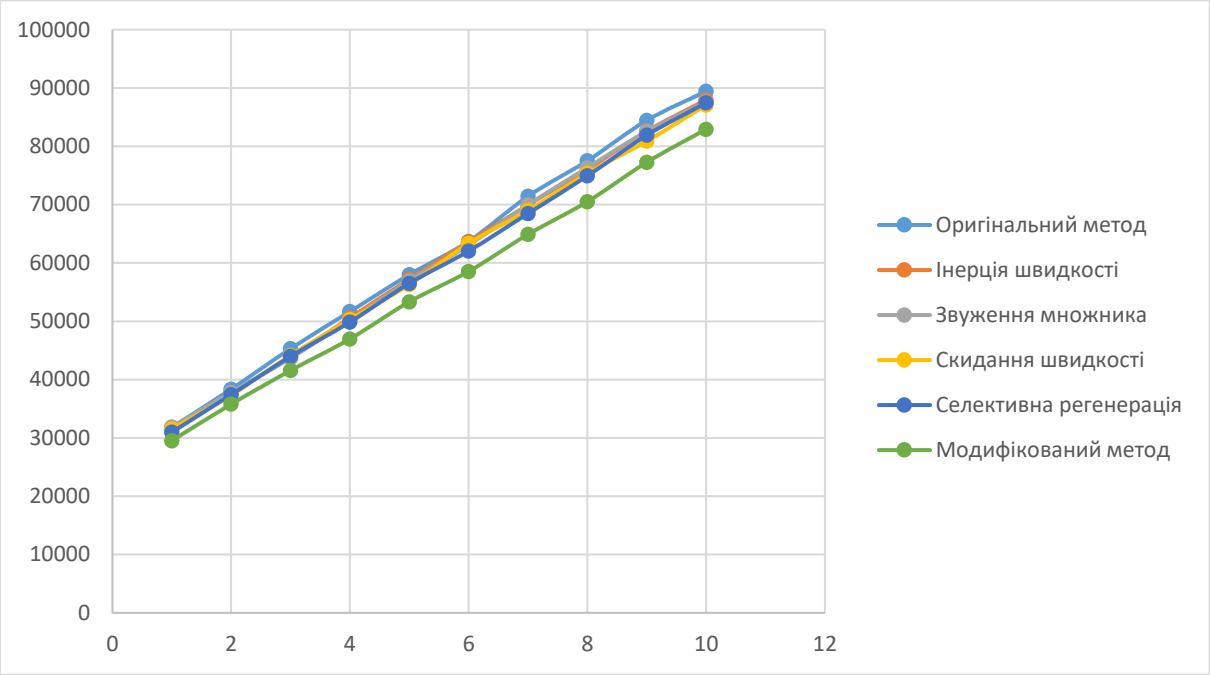


Схема 4.1.2 – Залежність усередненого часу виконання методу в секундах від номеру топології

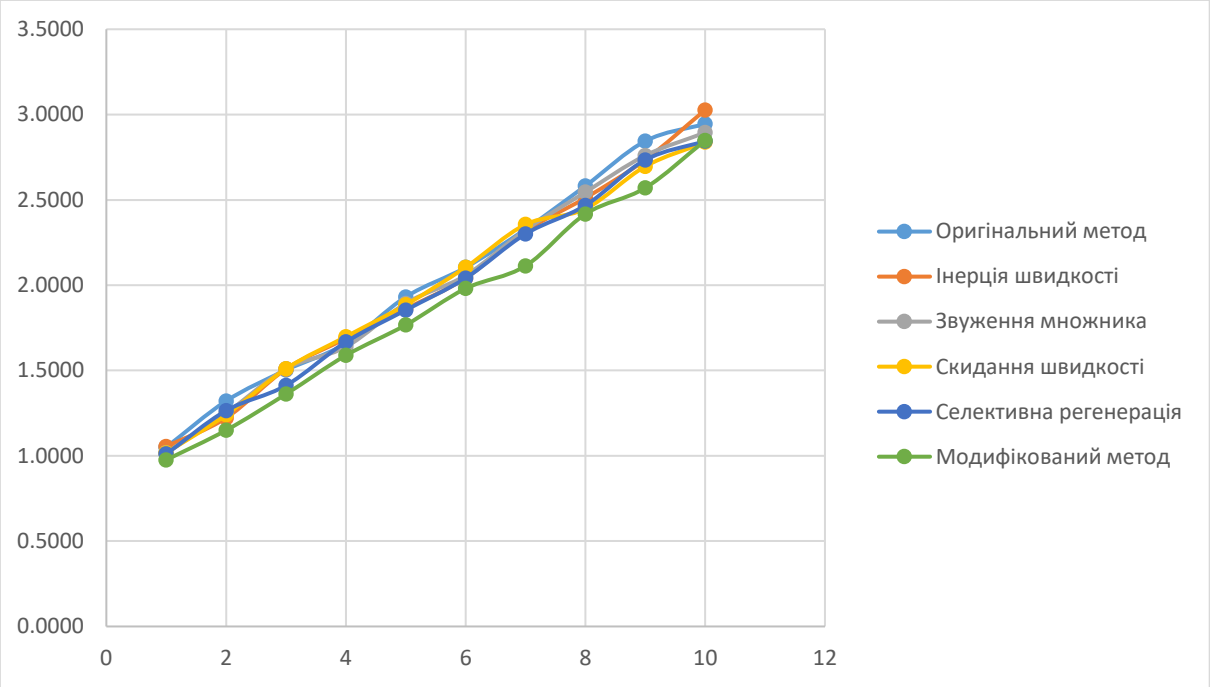
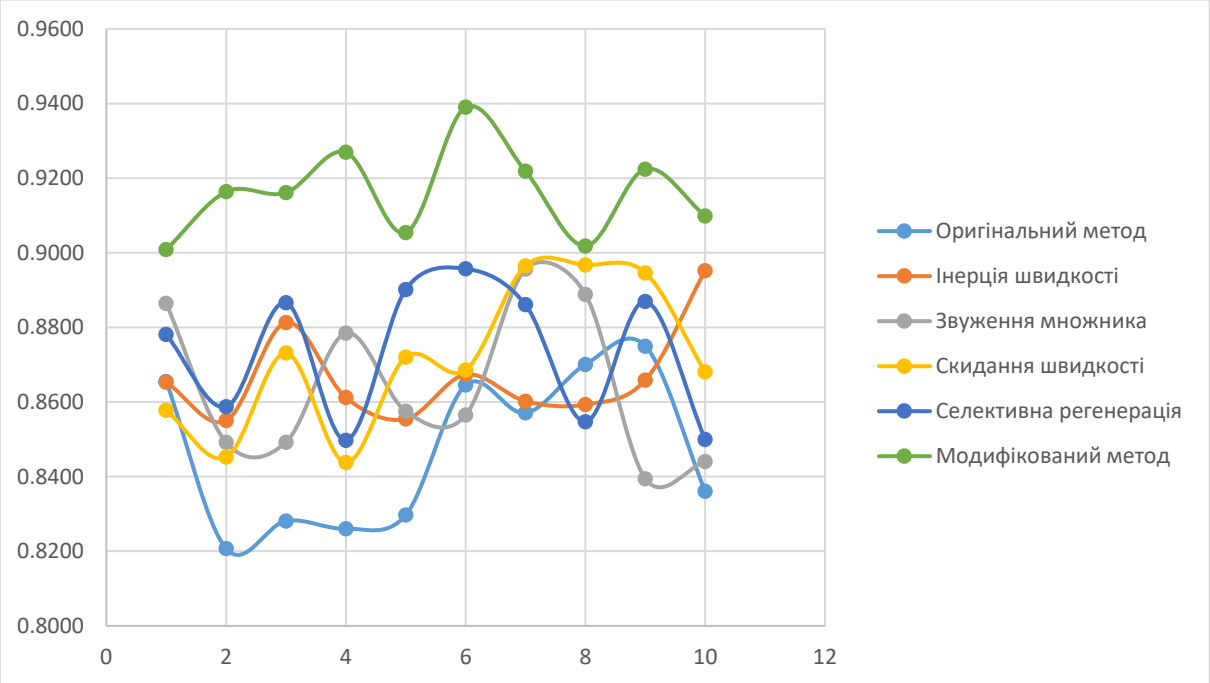


Схема 4.1.3 – Залежність усередненого значення показника успішності методу від номеру топології



4.2. Аналіз результатів

Аналізуючи отримані результати можемо зробити наступні висновки.

Перш за все бачимо, що кожна з запропонованих модифікацій методу рою частинок, в середньому, дає покращення в сенсі показника успішності та кількості ітерацій, хоча й не дуже значні за обсягами.

По-друге, всі запропоновані модифікації крім звуження множника мають приблизно той самий час виконання методу, хоча й знаходять розв’язок за меншу кількість ітерацій. Це пояснюється тим, що дані модифікації вимагають додаткових операцій та обчислень на кожній ітерації методу. Звуження множника, в свою чергу, виконує оновлення.

Якщо проводити більш детальний аналіз, то можна виявити, що модифікації методу покращують характеристики кількості ітерацій, часу виконання та показника успішності алгоритму на такі величини:

- Інерція швидкості ~ 2%.
- Звуження множника ~ 2%.
- Скидання швидкості ~ 2,5%.
- Селективна регенерація ~ 3%.
- Метод з усіма модифікаціями ~ 8%.

З даних показників можемо зробити висновок, що всі модифікації дають приблизно однаковий вклад в покращення характеристик методу.

Також скидання швидкості дозволяє отримати дещо кращі результати ніж інерція швидкості та звуження множника. Це можна пояснити тим, що скидання швидкості використовує реініціалізацію певних частинок рою, чим, можна так сказати виконує «відкидання» методу на якийсь час назад.

Також бачимо, що селективна регенерація має кращі за інші модифікації показники, навіть кращі за скидання швидкості. Це можна пояснити тим, що хоча скидання швидкості та селективна регенерація є

дещо подібними за своєю реалізацією, все ж селективна регенерація використовує більш «розумний» алгоритм відбору частинок а також використання параметрів їх когнітивної поведінки.

ВИСНОВКИ

В процесі написання магістерської дисертації було опрацьовано літературні джерела щодо поставленої задачі та мети розробки, досліджено існуючі алгоритми пошуку найкоротших шляхів в телекомунікаційних мережах та на графах, метаевристичні алгоритми, а також оригінальний метод рою частинок, запропоновано кілька його модифікацій та розроблено модифікований метод рою частинок для розв'язання задачі пошуку найкоротшого шляху в телекомунікаційній мережі, виконано порівняння результатів роботи оригінального та модифікованого методів, а також кожної із запропонованих модифікацій окремо при роботі з різними наборами даних.

Результатом магістерської дисертації є модифікований метод рою частинок, а також програмні засоби, що виконують реалізацію даних методів, які дозволяють перевірити запропоновану наукову гіпотезу на практиці. Розроблений модифікований метод рою частинок може використовуватись як для задачі оптимізації функції багатьох змінних в неперервному просторі, так і для задачі пошуку найкоротшого шляху в телекомунікаційній мережі.

З результатів тестування запропонованих модифікацій встановлено, що всі модифікації вносять долю покращення якісних характеристик методу в сенсі кількості ітерацій для збіжності, часу виконання та значення показника успішності відповідно до мети роботи.

В результаті розробки програмних засобів систематизовано знання технології .NET Core, мови програмування C#, а також роботи в середовищі розробки Microsoft Visual Studio.

Предметом подальших досліджень можуть бути модифікації методу, пов'язані зі застосуванням шумових метаевристик, а також введення додаткового члена в формулу переміщення – групи.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. MIT Press і McGraw-Hill. ISBN 0-262-03141-8. Секція 26.2, «The Floyd–Warshall algorithm», ст. 558–565 та секція 26.4, «A general framework for solving path problems in directed graphs», ст. 570–576.
2. Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2001). "Section 24.3: Dijkstra's algorithm". Introduction to Algorithms (Second ed.). MIT Press and McGraw–Hill. pp. 595–601. ISBN 0-262-03293-7.
3. Зорін Ю.М. Методичні вказівки до курсу «Комп’ютерні системи штучного інтелекту». Алгоритм бджолиної колонії. 2017 р.
4. Зорін Ю.М. Методичні вказівки до курсу «Комп’ютерні системи штучного інтелекту». Мурашині алгоритми. 2017 р.
5. Зорін Ю.М. Методичні вказівки до курсу «Комп’ютерні системи штучного інтелекту». Алгоритм зозулі. 2017 р.
6. Зорін Ю.М. Методичні вказівки до курсу «Комп’ютерні системи штучного інтелекту». Алгоритм кажанів. 2017 р.
7. Зорін Ю.М. Методичні вказівки до курсу «Комп’ютерні системи штучного інтелекту». Алгоритм світлячків. 2017 р.
8. Зорін Ю.М. Методичні вказівки до курсу «Комп’ютерні системи штучного інтелекту». Алгоритм рою частинок. 2017 р.
9. Sean Luke, 2009, Essentials of Metaheuristics, First Edition.
10. Kennedy, J.; Eberhart, R. (1995). "Particle Swarm Optimization". Proceedings of IEEE International Conference on Neural Networks. IV. pp. 1942–1948. doi:10.1109/ICNN.1995.488968.
11. Bonyadi, M. R.; Michalewicz, Z. (2017). "Particle swarm optimization for single objective continuous space problems: a review". Evolutionary Computation. 25 (1): 1–54. doi:10.1162/EVCO_r_00180.

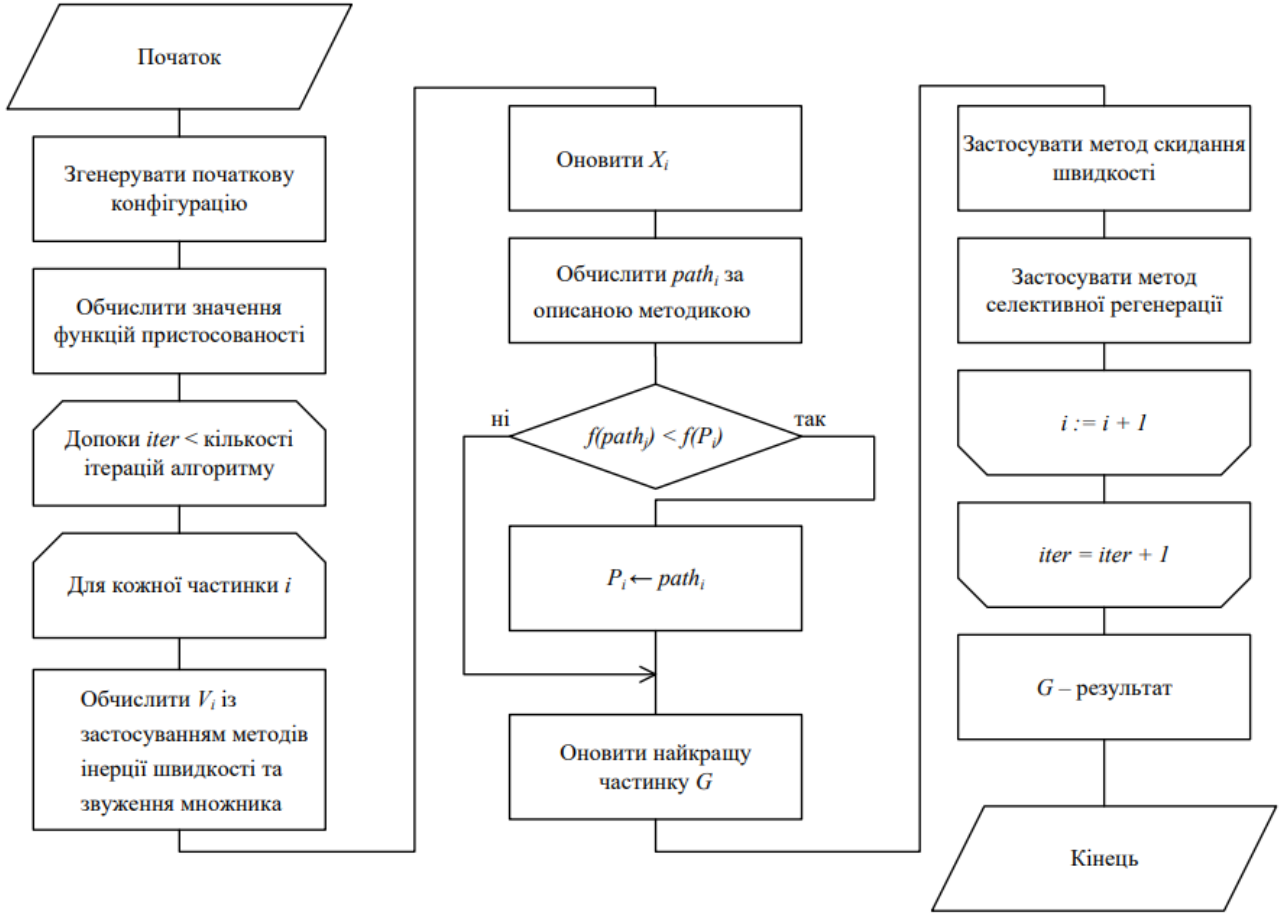
12. J. Inagaki, M. Haseyama, and H. Kitajima, “A genetic algorithm for determining multiple routes and its applications,” in Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS ’99), vol. 6, pp. 137–140, Orlando, Fla, USA, May-June 1999.
13. M. Gen, R. Cheng, and D. Wang, “Genetic algorithms for solving shortest path problems,” in Proceedings of the IEEE International Conference on Evolutionary Computation, pp. 401–406, Indianapolis, Ind, USA, April 1997.
14. Ammar W. Mohemmed and Nirod Chandra Sahoo. Efficient Computation of Shortest Paths in Networks Using Particle Swarm Optimization and Noising Metaheuristics. Discrete Dynamics in Nature and Society Volume 2007, Article ID 27383, 25 pages doi:10.1155/2007/27383
15. Fitness functions in evolutionary robotics: A survey and analysis (AFFG) (PDF), огляд функцій пристосованості, що використовуються в еволюційній робототехніці.
16. R. C. Eberhart and Y. Shi, “Comparing inertia weights and constriction factors in particle swarm optimization,” in Proceedings of the Congress on Evolutionary Computation (CEC ’00), vol. 1, pp. 84–88, La Jolla, Calif, USA, July 2000.
17. .NET Framework Developer Center on MSDN Library.
18. “Standard ECMA-335: Common Language Infrastructure (CLI)”. ecma-international.org (6 ed.). ECMA. June 2012.
19. Landwerth, Immo (4 December 2014). "Introducing .NET Core". .NET Framework Blog. Microsoft. Retrieved 17 April 2018.
20. Бондарчук М.Ю, Зорін Ю.М., Алгоритм рою частинок для знаходження найкоротшого шляху в графі, *Прикладна математика та комп’ютинг 2011 с.27-31*.

21. Бондарчук М.Ю, Зорін Ю.М., модифікований алгоритм рою частинок для пошуку найкоротших шляхів в телекомунікаційних мережах, *Прикладна математика та комп'ютинг 2018 с.49-53*.

ДОДАТКИ

Додаток 1. Копії графічних матеріалів

Схема модифікованого методу рою частинок



Застосування методу до задачі пошуку шляху телекомунікаційної мережі

// i – початковий вузол

// j – сусідній до i вузол

// n – кінцевий вузол

// 1 дорівнює початковому вузлу

// $A(i)$ – набір вузлів, що є сусідами вузла i

// $PATH(k)$ – частковий шлях на кроці декодування k

// p_j – відповідний пріоритет вузла j в частинці P (вектор X)

// N_∞ – задане велике число

$Particle\_Decoding(P)$

$i \leftarrow 1,$

$p_1 \leftarrow N_\infty,$

$k \leftarrow 0,$

$PATH(k) \leftarrow \{1\}$

$while\ \left(\{j \in A(i), p_j \neq N_\infty\} \neq \emptyset\right)$

$k \leftarrow k + 1$

$j \leftarrow \text{argmin}\{c_{ij} p_j | j \in A(i), p_j \neq N_\infty\}$

$i \leftarrow j$

$PATH(k) \leftarrow PATH(k) \cup \{i\}$

$p_i \leftarrow N_\infty$

$if\ i = n$

$return\ PATH(k)$

$end\ while$

$return$ Неправильний шлях

Модифікації методу рою частинок, спрямовані на покращення когнітивної поведінки рою

Цільова функція для мінімізації

$$\sum_{(i,j)\in E}^{h_{uj}} c_{ij},$$

де

$$\sum_{j:(i,j)\in E} h_{ij} - \sum_{j:(i,j)\in E} h_{ji} = \begin{cases} 1, i = s \\ -1, i = t, \\ 0, i \neq s, t \end{cases}$$

де h_{ij} – це 1, якщо ребро з'єднує вершини i та j на шляху та 0 інакше.

Інерція швидкості:

$$w_{iter} = w_{max} + (w_{max} - w_{min}) \frac{iter}{I}.$$

Звуження множника:

$$\chi = \begin{cases} 2\left(\left|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}\right|\right)^{-1}, \varphi = \varphi_p + \varphi_g > 4 \\ 1, \varphi = \varphi_p + \varphi_g \leq 4 \end{cases}.$$

Селективна регенерація: серед тих частинок, «відстань» яких є меншою за задане значення u повторно ініціалізувати $d\%$.

Формула зміни швидкості частинки:

$$v_{ij}(t+1) = \chi \left[w v_{ij}(t) + \varphi_p R_1 \left(p_{ij}(t) - x_{ij}(t) \right) + \varphi_g R_2 \left(p_{gj}(t) - x_{ij}(t) \right) \right]$$

Табличні результати тестування

Результати тестування оригінального алгоритму

Номер топології	Кількість вузлів	Кількість ітерацій	Час, секунд	SR
1	50	31807	1.0501	0.8655
2	60	38366	1.3212	0.8207
3	70	45313	1.5025	0.8281
4	80	51660	1.6585	0.8260
5	90	57996	1.9313	0.8298
6	100	63705	2.1043	0.8646
7	110	71444	2.3281	0.8571
8	120	77537	2.5821	0.8701
9	130	84450	2.8446	0.8750
10	140	89452	2.9458	0.8361

Результати тестування методу скидання швидкості

Номер топології	Кількість вузлів	Кількість ітерацій	Час, секунд	SR
1	50	31558	1.0177	0.8578
2	60	37317	1.2396	0.8452
3	70	44169	1.5103	0.8731
4	80	50284	1.6984	0.8438
5	90	56297	1.8853	0.8720
6	100	63357	2.1052	0.8686
7	110	69015	2.3560	0.8965
8	120	75378	2.4476	0.8968
9	130	80904	2.6964	0.8946
10	140	87094	2.8381	0.8680

Результати тестування методу інерції швидкості

Номер топології	Кількість вузлів	Кількість ітерацій	Час, секунд	SR
1	50	31604	1.0545	0.8653
2	60	37656	1.2221	0.8550
3	70	43881	1.5087	0.8813
4	80	50597	1.6871	0.8612
5	90	57207	1.8616	0.8554
6	100	63587	2.0419	0.8674
7	110	69950	2.3170	0.8602
8	120	76037	2.5115	0.8593
9	130	82622	2.7353	0.8659
10	140	87964	3.0272	0.8952

Результати тестування селективної регенерації

Номер топології	Кількість вузлів	Кількість ітерацій	Час, секунд	SR
1	50	30962	1.0104	0.8781
2	60	37420	1.2649	0.8587
3	70	43971	1.4124	0.8866
4	80	49857	1.6670	0.8498
5	90	56511	1.8543	0.8901
6	100	62052	2.0416	0.8957
7	110	68488	2.3005	0.8861
8	120	74938	2.4679	0.8547
9	130	81913	2.7342	0.8870
10	140	87461	2.8446	0.8500

Результати тестування методу звуження множника

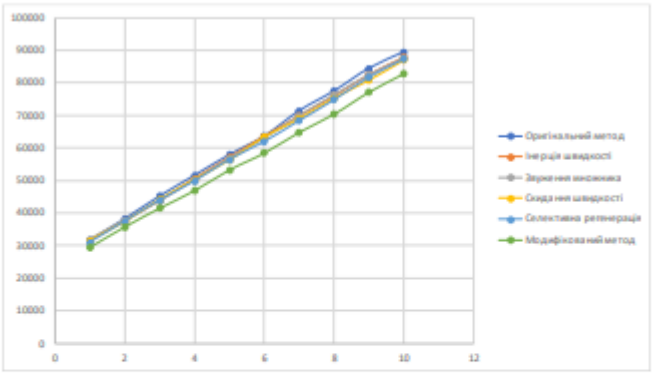
Номер топології	Кількість вузлів	Кількість ітерацій	Час, секунд	SR
1	50	31849	1.0181	0.8865
2	60	37816	1.2491	0.8492
3	70	43688	1.5105	0.8492
4	80	50221	1.6383	0.8785
5	90	56905	1.8939	0.8575
6	100	63077	2.0576	0.8565
7	110	69981	2.3240	0.8956
8	120	76349	2.5454	0.8889
9	130	82568	2.7600	0.8395
10	140	87655	2.8943	0.8441

Результати тестування модифікованого алгоритму

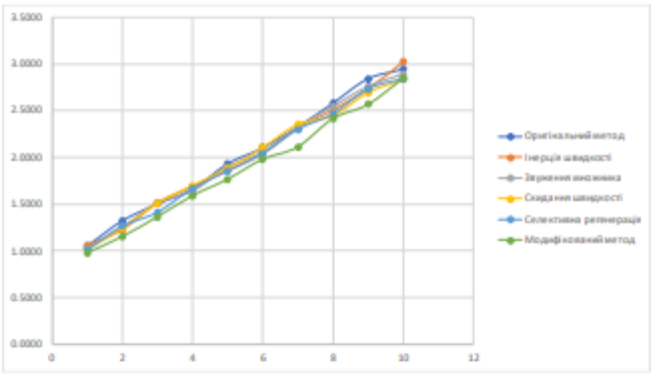
Номер топології	Кількість вузлів	Кількість ітерацій	Час, секунд	SR
1	50	29524	0.9761	0.9008
2	60	35790	1.1499	0.9164
3	70	41565	1.3619	0.9161
4	80	46972	1.5886	0.9269
5	90	53338	1.7668	0.9054
6	100	58546	1.9807	0.9390
7	110	64916	2.1133	0.9218
8	120	70521	2.4173	0.9018
9	130	77269	2.5703	0.9224
10	140	82942	2.8482	0.9099

Графічні результати тестування

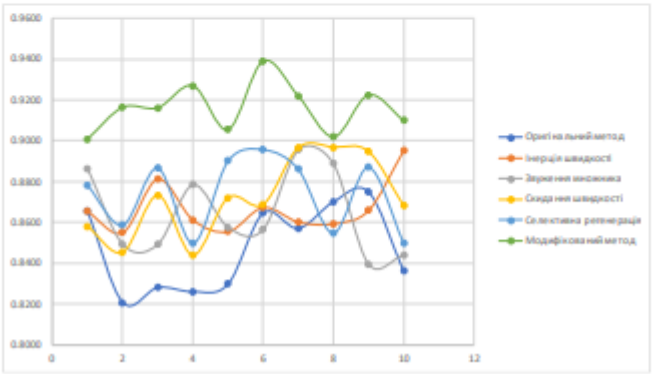
Залежність усередненої кількості ітерацій методу від номеру топології



Залежність усередненого часу виконання методу в секундах від номеру топології



Залежність усередненого значення показника успішності методу від номеру топології



Модифікований метод рою частинок

Згенерувати початкову конфігурацію (для кожної частинки):

Виконати довільну ініціалізацію X_i

Виконати довільну ініціалізацію V_i

Обчислити $path_i$ за описаною методикою

Обчислити значення функції пристосованості $f(X_i)$

$P_i \leftarrow \infty$

$G \leftarrow \infty$

$iteration\_count \leftarrow 0$

// max_$iteration$ – максимальна кількість ітерацій

while ($iteration\_count$ < max_$iteration$)

для кожної частинки i

Обчислити V_i із застосуванням методів інерції швидкості та звуження множника

Оновити X_i

Обчислити $path_i$ за описаною методикою

Обчислити значення функції пристосування $f(path_i)$

if $f(path_i) < f(P_i)$

$P_i \leftarrow path_i$

if $f(P_i) < f(G)$

$G \leftarrow P_i$

Застосувати метод скидання швидкості

Застосувати метод селективної регенерації

$iteration\_count \leftarrow iteration\_count + 1$

end while

return G